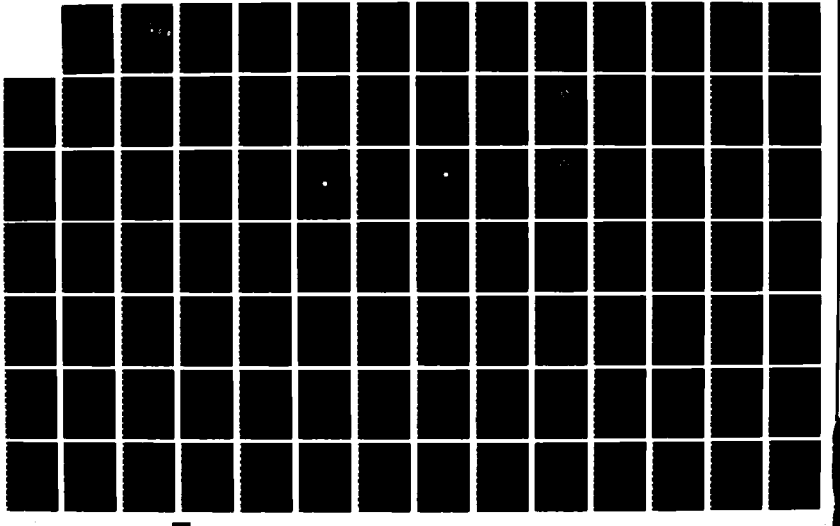
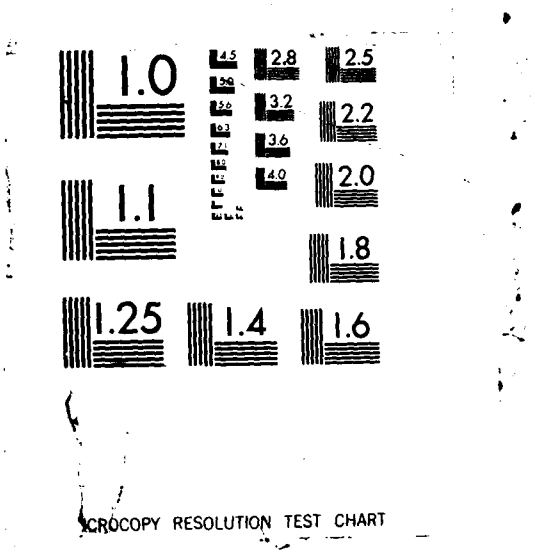
AD-A175 327 USE OF THE RADIO SHACK TRS-80 MODEL 100 COMPUTER FOR COMBAT MODELING(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S H CARY SEP 86 1/2

UNCLASSIFIED F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
CROCOPY RESOLUTION TEST CHART

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

USE OF THE RADIO SHACK TRS-80 MODEL 100
COMPUTER FOR COMBAT MODELING

by

Steven H. Cary

September 1986

Thesis Advisor: James N. Eagle

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) Code 55	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
USE OF THE RADIO SHACK TRS-80 MODEL 100 COMPUTER FOR COMBAT MODELING

12 PERSONAL AUTHOR(S)
Cary, Steven H.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986, September	117

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	M100, detection, simulation, matrix algebra, Kalman, filter, Lanchester, numerical integration, geometric programming

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The primary purpose of this thesis is to demonstrate some principles of combat modeling using programs for the Radio Shack TRS-80 Model 100 computer. In addition to the combat modeling, the thesis includes several utility programs for the M100 of interest to students of operations analysis.

The combat modeling programs include an antisubmarine warfare (ASW) detection simulation, a Kalman filter, and a Lanchester differential equation simulation. The utility programs include a matrix algebra program, a numerical double integration program using Simpson's Rule and the Romberg integration algorithm, and a geometric programming program for zero degree of difficulty problems. The integration program is also written as a subroutine that can be included in other programs. The matrix algebra program includes a simultaneous linear equation solving subroutine which can be used in other programs.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL James N. Eagle	22b TELEPHONE (Include Area Code) (408) 646-2654 / 22c OFFICE SYMBOL Code 55Er

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

19. ABSTRACT

All programs are written in M100 BASIC. Documentation includes an explanation of the input required, the output produced, and the components of each program, and sample problems. The chapter on geometric programming includes a tutorial on the mathematical basis for that technique.

Approved for public release; distribution is unlimited.

Use Of The Radio Shack TRS-80 Model 100 Computer
For Combat Modeling

by

Steven H. Cary
Captain, United States Army
B.S., Kansas State University, 1974

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1986

Author: Steven H. Cary

Approved by: James N. Eagle, Thesis Advisor

J. D. Esary, Second Reader

Peter Purdue, Chairman,
Department of Operations Research

Kneale T. Marshall,
Dean of Information and Policy Sciences

ABSTRACT

The primary purpose of this thesis is to to demonstrate some principles of combat modeling using programs for the Radio Shack TRS-80 Model 100 computer. In addition to the combat modeling, the thesis includes several utility programs for the M100 of interest to students of operations analysis.

The combat modeling programs include an antisubmarine warfare (ASW) detection simulation, a Kalman filter, and a Lanchester differential equation simulation. The utility programs include a matrix algebra program, a numerical double integration program using Simpson's Rule and the Romberg integration algorithm, and a geometric programming program for zero degree of difficulty problems. The integration program is also written as a subroutine that can be included in other programs. The matrix algebra program includes a simultaneous linear equation solving subroutine which can be used in other programs.

All programs are written in M100 BASIC. Documentation includes an explanation of the input required, the output produced, and the components of each program, and sample problems. The chapter on geometric programming includes a tutorial on the mathematical basis for that technique.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

The NPS Operations Research Department has made Radio Shack TRS-80 Model 100 computers available to a limited number of students. Experience has shown that these students have made relatively limited use of these computers. The main reasons seem to be:

- This computer uses the BASIC programming language. NPS OA students do not receive instruction in this language but are required to learn FORTRAN and APL. Although FORTRAN and BASIC have similar structures, most OA students believe they do not have time to learn a third computer language.
- Only a few M-100 programs are currently available at NPS that directly relate to course work.
- Writing and debugging programs for the M100 can take a considerable amount of time. Many OA students believe that time would be more usefully spent pursuing other approaches to their studies.
- The M100 has thus far not been distributed to all OA students. Therefore, professors have not been able to require students to use the M100. It has been relegated to "nice to have" status instead of being included as an essential teaching tool.

B. GENERAL

The purpose of this paper is to develop a collection of programs in BASIC for the M100 that students can use in the combat modeling courses of the OA curriculum. The programs make extensive use of subroutines which allow students to run programs "off the shelf" or build their own programs from the subroutines. The programs are extensively documented so that students who learn BASIC can use the printed programs as study aids to understand the algorithms involved. Some of the programs are tutorial.

In testing situations professors are often limited to problems that are arithmetically very simple. If programs for the M100 were available, professors would be able to give more intricate test problems without placing undue emphasis on manual arithmetic calculation.

Additionally, when students leave NPS, they will be able to take with them a series of programs with which they have grown familiar during their course of study.

II. FEATURES COMMON TO MORE THAN ONE PROGRAM

A. GENERAL CHARACTERISTICS OF THE MODEL 100

The M100 is a versatile and very portable computer. As provided to Naval Postgraduate School students, it has either 24K or 32K of RAM for storage of variables during program execution and for storage of programs and other files. Programs and files remain active while the computer is turned off. There is an internal 300 baud modem which facilitates transfering programs to other computers for storage. The eight line LCD screen limits the graphics display capability of the M100. However, output in character form can be written to RAM files and reviewed after program execution is complete. The version of BASIC used in the M100, creation of text files, use of the modem, etc. are explained in Reference 1. This thesis assumes that readers are somewhat familiar with Reference 1.

Although the programs have statements which print intermediate and final results to the screen, the operator may wish to check the status of a variable that is not printed by the program. To do this

- Stop the program by depressing the SHIFT and BREAK keys simultaneously.
- Type a BASIC statement to print the desired variables and hit the ENTER button.
- Start the program again at the place it stopped by typing CONT and hitting the ENTER button.

Most of the programs in this thesis do not include statements to end the program. This is because the programs cycle back to the start of the program allowing the operator to enter a new set of parameters without restarting the program. To end the program

- Depress the SHIFT key and the BREAK key at the same time.
- To rerun the program from the start, depress the F4 key.
- To return to the main menu, depress the F8 key.

B. COMMON TERMINOLOGY

BASIC variables are refered to in the text using capital letters. Since M100 BASIC only differentiates between variables based on the first two letters of the variable name, most BASIC variables in the following programs are combinations of

two capital letters, e.g. BA, CR, FT. Names of matrices are also specified using capital letters. BASIC permits matrices to be dimensioned using variables. If, for instance, AB=3 and AC=4, then the BASIC statement DIM ZZ(AB,AC) would dimension a three by four matrix named ZZ. When capital letters are used inside the parentheses, the size of the matrix is being specified. When a small letter is used inside the parentheses, a particular element of the matrix is being specified. For example, ZZ(i,j) refers to the element of ZZ in the ith row and jth column. When a matrix has more than one dimension, the first number from the right is refered to as the column number and the second number from the right as the row number.

When the mathematical theory behind a program is discussed, the variables used will be small letters or Greek letters.

C. INPUT FILES

All of the programs presented in this thesis permit data to be entered from a text file. These text files must be created using the M100's text editor before the program is run. The name of the text file for each program is similar to the name of the program it supports and is given in the documentation for each program under the section titled, "Input". The contents of the input file are also explained in the appropriate "Input" section.

Numbers in input files must be separated by a space, comma, or return. A comma and a return may not be placed together without a number between them. Otherwise, the M100 will enter an extra zero at that point. A comma and a return together also cause data elements which follow to be in the wrong places in their matrices and/or wrong numbers to be read as matrix dimensions. Do not put blank lines[1] in the data for the same reason.

D. FORMULA TOKENIZATION

There are some programs which must be adapted to use different equations at the same point in the program depending on the application. For example, the detection theory simulation in Chapter 3 must be able to handle many functions for the probability of detection of a sensor. When the function needs to be changed, the operator may:

[1] Two returns together.

- Stop the program, call the BASIC editor for the lines that need to be changed, and restart the program after the function has been edited, or
- Use a tokenization routine to change the function without stopping the program.

This section describes a subroutine (see Figure 2.1) which performs that tokenization.

```
1400 'Tokenize DF
1410 B$="DF="+DF$+CHR$(0)
1450 'Tokenize/execute B$
1451 B0=VARPTR(B$):B1=PEEK(B0+1)+256*PEEK(B0+2):CALL1606,0,B1
1455 CALL2499,0,63105:RETURN
```

Figure 2.1 Formula Tokenization Section.

That subroutine is taken from [Ref. 2:pp. 58-60]. Tokenization converts a string variable into an executable BASIC statement. In the example in Figure 2.1, the right hand side of the function equation was stored as a string variable, DFS before this subroutine was called. For example, if the equation was DF=X+Y, then DFS is the string, "X+Y". Line 1410 adds an equal sign and appropriate left hand side to DFS to form BS, a complete assignment statement in string format. In the example above line 1410 adds "DF=" to DFS to form BS. Line 1451 computes memory location, B1, of BS and calls the machine level subroutine in the M100 read only memory (ROM) beginning at memory location 1606. Subroutine 1606 converts the BASIC keywords in BS into their one byte tokens and then stores them in executable form at memory location 63105. Line 1455 calls the machine level subroutine beginning at memory location 2499 which executes the statement stored at memory location 63105.

III. DETECTION SIMULATION PROGRAM

A. GENERAL

The problem addressed by this program is estimating the probability of detecting a target whose location is given by a bivariate normal distribution when the location of each detecting sensor also has a separate BVN distribution. The target distribution is BVN(0,0,σ^2_x,σ^2_y,$\rho_{x,y}$). The location of each sensor, S_i, is distributed BVN(μ_{u_i}, μ_{v_i}, $\sigma^2_{u_i}$, $\sigma^2_{v_i}$, ρ_{u_i,v_i}). All distribution parameters are stored in an input text file. The program models two sensor types:

- A Deterministic Sensor. This sensor has detection probabilities of 1, 0 and 1 in three concentric circular bands around the sensor's location. An example of this type of sensor would be a sonobouy which detects all targets out to range r_1, detects no targets between ranges r_1 and r_2, detects all targets in a convergence zone between r_2 and r_3, and detects no targets beyond r_3 where $0 \leq r_1 \leq r_2 \leq r_3$.
- A Probabilistic Sensor. This sensor has a continuous, radially symmetric detection pattern. The probability of detection, P_d, is a function of range and may also be a function of one or more other parameters. The default function, D(r,b), for calculating P_d is a Carlton function where b is a scaling parameter (See Equation 3.1). Figure 3.1 shows graphs of several Carlton detection functions.

$$P_d = D(r,b) = e^{-r^2/2b^2} \qquad \text{(eqn 3.1)}$$

For either type of sensor the program calculates the overall probability of detection using D(r). The deterministic sensor portion of the program can also be used to calculate a "cookie cutter" approximation of the actual detection function. The cookie-cutter approximation has the form $D(r)=1$ when $r \leq r_o$ and $D(r)=0$ when $r > r_o$, for some specified detection range, r_o. The radius, r, of the cookie-cutter approximation is the lethal radius of the actual detection function for a sensor. *Lethal radius* is a term borrowed from the damage functions of firing theory. It is used here to help readers who are familiar with firing theory, not because being detected by a

sensor is inherently lethal. The lethal radius is a scalar measure of a sensor's detection ability and is computed using Equation 3.2. See [Ref. 3:pp. 12-15].

$$\text{Lethal Radius} = \left[2\int_0^{\infty} rD(r)\,dr\right]^{.5} \qquad \text{(eqn 3.2)}$$

To calculate a cookie-cutter approximation, the lethal radius of D(r) must be computed off line and included as a sensor parameter.[2] For example, the lethal radius of the default Carlton detection function is $2^{.5}b$.

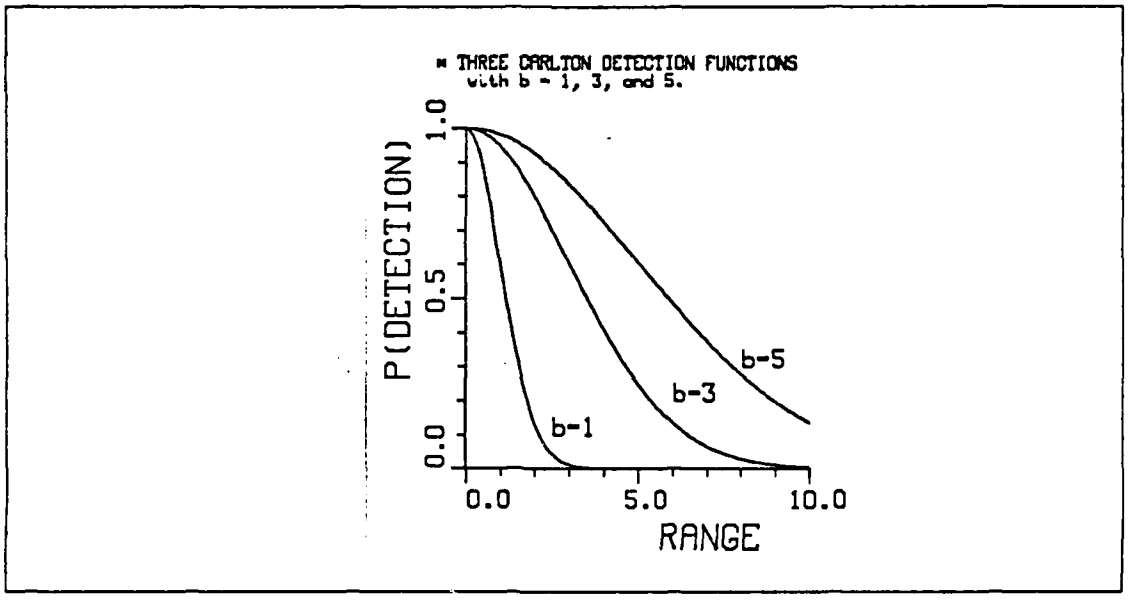


Figure 3.1 Graph of Carlton Detection Functions.

Closed form solutions are available for some specific cases that can also be computed with this program, e.g. a group of sensors with Carlton detection functions at fixed locations. The program will verify these closed form solutions. However, it will also estimate probability of detection for problems without closed form solutions, e.g. a group of cookie-cutter sensors in which each sensor has a different BVN location distribution.

---

[2]See the second example in the section of example problems at the end of this chapter.

The program uses a Monte Carlo simulation to estimate P_d. Because the Monte Carlo simulation is time consuming, faster numerical integration approximations are also available for the deterministic sensor. *However, the particular numerical integration technique used in this program will not produce correct results if sensor areas of coverage overlap.* More sophisticated numerical integration techniques are available for special cases of the probabilistic sensor, but are not included in this program.

B. EXPLANATION OF VARIABLES

- AL is the probability that the computed confidence level does not contain the true probability of detection for the associated standard normal CDF value.
- A2(6,6) is the intermediate calculation matrix for the Romberg integration subroutine.
- B0 and B1 are the memory location parameters of BS.
- BS is the string that holds the entire equation for DF in the tokenization process.
- D1 and D2 are intermediate calculation values.
- DF is the value of the detection function.
- DFS is the string that hold the right hand side of the equation for DF.
- F is the value of the function being integrated at a specific point.
- F1 is, in the data input section (lines100-150), the selection variable indicating whether all sensors have the same parameters. If F1 = 1, then sensor parameters, other than the five for location, will be listed only once: after the location parameters of S_1. If F1 = 0, then values of all parameters for all sensors must be entered explicitly in DSIN.DO. In the rest of the program, beyond line 200, F1 is a flag determining whether the calculation is based on the deterministic (F1 = 1) or the probabilistic (F1 = 2) sensor.
- F2 is the selection variable for whether all sensors locations are distributed with $\sigma_x = \sigma_y = 0$. 1 - yes; 2 - no.
- F3 is the selection variable for whether a Monte Carlo simulation (F3 = 1) or a numerical integration (F3 = 2) is used.
- H is the radius of circular limits of integration.
- I, I1, I2 are loop counters.
- IN is the value of the integral from a numerical approximation.
- J1 through J9 are loop counters.
- NS is the number of sensors.
- NP is the number of parameters for each sensor in addition to location.

- NR is the number of repetitions in a Monte Carlo simulation.
- PD is the probability of detection.
- RF is $(1\text{-}RH^2)^{.5}$.
- RH is the correlation between components of the BVN target location distribution.
- S1 and S2 are standard deviations of the components of the target location distribution.
- TE is a temporary storage variable.
- T1S and T2S are times at beginning and end of a calculation.
- T1 is the time elapsed during a calculation.
- V1 and V2 are variances of the components of the target location distribution.
- XS and XT are specific values of the first component, i.e. the mean X position, of the BVN distributions of a sensor or the target respectively.
- X(NS,5+NP) is the parameter matrix for sensors. There is one row per sensor and one column per parameter. Columns one through five are for the means, μ_{u,S_i} and μ_{v,S_i}, the standard deviations, σ_{u,S_i} and σ_{v,S_i}, and the correlation, ρ_{S_i}, of the location of each sensor, S_i. These parameters are in the same coordinate system as the target location distribution.
- YS and YT are specific values of the second component, i.e. the mean Y position, of the BVN distributions of a sensor or the target respectively.
- Z1 and Z9 are selection variables.

C. INPUT

1. Input File

Before this program is run, a text file, DSIN.DO, must be created to hold the input parameters. DSIN.DO will contain the following variables in the following order:

- NS, NP, S1, S2, RH, F1
- The sensor location/parameter(s) matrix, X(NS,NP+5).

An example of a input file is shown in Figure 3.2 along with a graph of the corresponding sensor coverage areas. Entries in the first line of Figure 3.2 indicate that there are two sensors with three parameters each, that the target location distribution is BVN(0, 0, 625, 625, 0), and that all sensors have the same parameters. The second line is the parameter set for the first sensor. There is no variance in the location of

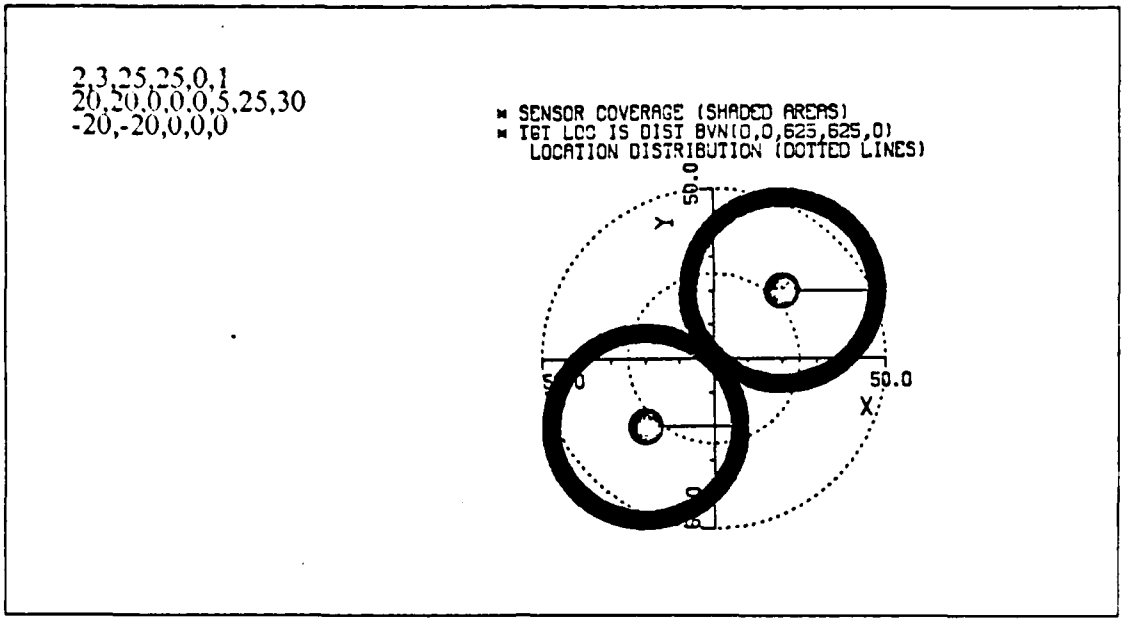


Figure 3.2 Example of Input File, DSIN, And Graph Of Sensor Coverage.

either sensor. The first sensor is located at (20,20) and has additional parameters 5, 25, and 30. If this input file represents sensors that have a deterministic detection functions, then the sensors detect all targets within a distance of 5 and within a circular band from 25 to 30 and do not detect targets outside of these bands. Both sensors have the same parameters other than location, as indicated by the last element of the first line being one, the last line contains only the location of the second sensor. If one or more sensors had different parameters (other than location), then the last element in the first row would have been zero. Also, the three nonlocation parameters would have been specified for all sensors. Although the parameters for each sensor may be different, the detection function for each sensor must be the same. For example, the program does not allow a mixture of sensors with deterministic and probabilistic detection functions.

2. Interactive Input

After the program has read the input file, the operator will be prompted to specify:

- Whether a deterministic or a probabilistic detection function will be used.
- Whether the variance in target location is or is not equal to zero.
- Whether a Monte Carlo simulation or numerical approximation will be used.

If the probabilistic detection function is used, the operator may interactively change the detection function. If the Monte Carlo simulation is used, the operator will be prompted to specify the number of repetitions, NR, in the simulation.

D. OUTPUT

The program prints the probability of detection, PD, to the screen of the M100. If the computation was a Monte Carlo simulation, the program asks the operator to specify an α[3] for the confidence interval on the probability of detection. Permissible α's are .10, .05, and .01. The program computes the lower and upper limits of the confidence interval based on a normal approximation to the binomial distribution. The expression for these limits is shown in Equation 3.3.

$$PD \pm Z_{1-\alpha/2}[PD(1-PD)/NR]^{.5} \qquad \text{(eqn 3.3)}$$

$Z_{1-\alpha/2}$ is the point on a standard normal distribution at which the CDF has a value of $1-\alpha/2$.

E. EXPLANATION OF PROGRAM COMPONENTS

A complete program listing is at Appendix A.

1. Initialization/Data Input, Figure 3.3

Lines 100-115 print the program title and open the input file, DSIN.

Line 120 reads the first line of DSIN and calculates the variances of the marginal distributions of the target location distribution. It also dimensions X(NS,5+NP) and A2(6,6).

Lines 125-135 read the lines of DSIN that specify parameters for individual sensors, S_i, i=1,2,...,NS. Line 125 reads all parameters for S_1. If some S_i have different nonlocation parameters, i.e. F1=0, then line 128 reads in 5+NP parameters for S_i where i=2,...,NS. If all sensors have the same nonlocation parameters, i.e. F1=1, then lines 132-135 read the five location parameters for each sensor. They also make nonlocation parameters of S_i, where i = 2,3,4,...,NS, equal to the nonlocation parameters of S_1.

---

[3] α is the probability that the computed confidence level does not contain the true probability of detection.

```
100 CLS:PRINT"":PRINT"          DETECTION SIMULATION":FORI=1TO400:NEXTI
110 'Input/Initialization
115 OPEN"DSIN"FORINPUTAS1
120 INPUT#1,NS,NP,S1,S2,RH,F1:V1=S1*S1:V2=S2*S2
121 DIMX(NS,5+NP),A2(6,6),T1(3)
125 FORI1=1TO5+NP:INPUT#1,X(1,I1):NEXTI1
126 IFNS=1THEN140
127 IFF1=1THEN132
128 FORI1=2TONS:FORI2=1TO5+NP:INPUT#1,X(I1,I2):NEXTI2:NEXTI1:GOTO140
132 FORI1=2TONS:FORI2=1TO5:INPUT#1,X(I1,I2):NEXTI2:IFNP=0THEN135
134 FORI2=6TO5+NP:X(I1,I2)=X(1,I2):NEXTI2
135 NEXTI1
140 RF=SQR(1-RH^2)
150 DF$="EXP(-((XT-XS)^2+(YT-YS)^2)/(2*X(J2,6)^2))"
```

Figure 3.3 Initialization and Data Input Section.

Line 140 calculates RF, an intermediate value used in later integration calculations.

Line 150 assigns the right hand side of the Carlton equation to DFS as the default equation for calculating the detection function value, DF.

2. **Method Selection Section, Figure 3.4**

```
200 '*** Simulation Selection Section ***
201 CLS:PRINT"Is the Detection function:"
203 PRINT" 1. Deterministic":PRINT" 2. Probabilistic"
205 INPUT"Enter 1 or 2:";F1
210 CLS:PRINT"Are Sensor Locations:"
212 PRINT" 1. Always At Aim Point":PRINT" 2. Distributed BVN Around Aim Point"
214 INPUT"Enter 1 or 2:";F2
215 IF(F1=2ORF2=2)THENF3=1:GOTO230
220 CLS:PRINT"":PRINT"Is the Calculation:"
222 PRINT" 1=Monte Carlo Simulation":PRINT" 2=Numerical Approximation"
224 INPUT"Enter 1 or 2";F3
230 TIME$="00:00:00":IF F1=1THENGOSUB300ELSEGOSUB500
250 GOTO200
```

Figure 3.4 Method Selection Section.

Lines 200-250 assign values to:

- F1. If F1 = 1, then calculations are done for a deterministic sensor. If F1 = 2, then calculations are done for a probabilistic sensor.

- F2. If F2 = 1, then the sensor is always located at its aiming point, i.e. $\sigma^2_x = \sigma^2_y = 0$. If F2 = 2, then σ^2_x or σ^2_y is not equal to 0.
- F3. If F3 = 1, then a Monte Carlo simulation is conducted. If F3 = 2, then a numerical approximation is calculated.

Line 230 sets the M100's clock to zero to time the calculation. It also branches to the appropriate subroutine depending upon which type of sensor is specified by F1.

3. **Deterministic Sensors, Lines 300-360**

```
300 'Deterministic Sensor Subroutine
305 IFF3=1THENGOSUB310ELSEGOSUB350
306 RETURN
```

Figure 3.5 Decision Logic For Deterministic Sensors.

Based upon the value of F3, lines 300-306 (see Figure 3.5) branch to subroutines to conduct the calculations using a Monte Carlo simulation or a numerical approximation.

a. Actual Detection Function, Lines 310-360

(1) *Monte Carlo Simulation, Figure 3.6.* Line 315 calls subroutine 900 which prompts the operator for the number of repetitions, NR.

```
310 'Monte Carlo of Deterministic Sensor
315 GOSUB900
320 PD=0:FORJ1=1TONR:PRINT.241,"Repetition:";J1:GOSUB600:FORJ2=1TONS
323 IFF2=2THENXS=X(J2,1):YS=X(J2,2):GOTO325
324 GOSUB612
325 T1=SQR((XS-XT)^2+(YS-YT)^2)
330 IFT1<=X(J2,6)THENPD=PD+1:GOTO335
332 IFT1>=X(J2,7)ANDT1<=X(J2,8)THENPD=PD+1:GOTO335
334 NEXTJ2
335 NEXTJ1:PD=PD/NR:GOSUB950:RETURN
```

Figure 3.6 Monte Carlo Simulation.

For repetitions one through NR, lines 320-335 generate a target location from the BVN distribution. They then check whether that location lies within

the circular detection bands around each sensor, S_i, i = 1,2,...,NS. PD counts the number of repetitions for which the target is in at least one detection band. The Monte Carlo simulation functions accurately even if detection bands of various sensors overlap because it does not double count if a target is detected by more than one sensor. When all repetitions are completed, the counter, PD, is divided by NR to produce an estimate of the probability of detection. Finally, output subroutine 950 is called.

(2) *Numerical Approximation, Figure 3.7.* The volume[4] under the target location distribution is calculated for the detection bands of each S_i. This volume is the sensor's probability of detection. *Overlapping detection bands are not permitted with this numerical approximation* because the program would double count the overlapping volumes. Subroutine 1200 conducts the integration. Probability of detection is the volume under the target location distribution that is within the coverage area of any S_i.

```
350 'Numeric/Deterministic Subroutine
355 PD=0:FORJ2=1TONS:H=X(J2,6):GOSUB1200:PD=PD+IN
356 H=X(J2,8):GOSUB1200:PD=PD+IN
357 H=X(J2,7):GOSUB1200:PD=PD-IN:NEXTJ2
360 GOSUB950:RETURN
```

Figure 3.7 Numerical Approximation.

4. **The Probabilistic Sensor, Figure 3.8**

Lines 502-503 print a header and call subroutine 1300 which permits the detection function to be modified. Line 503 also calls the subroutine in which the operator specifies the number of Monte Carlo repetitions.

For each repetition, lines 520-530 generate a target location from the BVN target location distribution. Then, for each S_i they calculate the value of the sensor's detection function using subroutine 1410 and generate a uniform random number between zero and one. If that random number is less than or equal to S_i's detection function value at that location, then S_i detected the target and PD is augmented by

[4]This volume, although it is computed and described as a volume in this section, is not a true volume. This is because although the X and Y axes of the BVN are distances, the Z axis is a probability, i.e. dimensionless. Therefore the result is really an area, not a volume.

```
500 'Probabilistic Detection Function
502 CLS:PRINT"Default Detection Function Is Carleton."
503 GOSUB1300:GOSUB900
520 PD=0:FORJ1=1TONR:PRINT.241,"Repetition:";J1:GOSUB600:FORJ2=1TONS
521 IFF2=2THENXS=X(J2,1):YS=X(J2,2):GOTO523
522 GOSUB612
523 GOSUB1410:IFRNS(1)<=DFTHENPD=PD+1:GOTO526
524 NEXTJ2
526 NEXTJ1:PD=PD/NR
530 GOSUB950:RETURN
```

Figure 3.8 The Probabilistic Detection Function.

one. If one sensor detects the target, the program moves directly to the next repetition. *Therefore, the Monte Carlo simulation functions accurately even if detection bands of various sensors overlap because it does not double count a target that is detected by more than one sensor.* When all repetitions are completed, the counter, PD, is divided by NR to produce an estimate of the probability of detection. Finally, output subroutine 950 is called.

5. **Generating A BVN Random Variable, Lines 600-606**

```
600 '***Generate BVN RV***
602 U1=RND(1):U2=RND(1):TE=SQR(-2*LOG(U1))
604 XT=TE*COS(6.2831853*U2):YT=RH*XT+RF*TE*SIN(6.2831853*U2)
606 XT=XT*S1:YT=YT*S2:RETURN
612 U1=RND(1):U2=RND(1):TE=SQR(-2*LOG(U1))
614 XS=TE*COS(6.2831853*U2):YS=X(J2,5)*XT+(1-X(J2,5)*2)^.5*TE*SIN(6.2831853*U2)
616 XS=X(J2,1)+XS*X(J2,3):YS=X(J2,2)+YS*X(J2,4):RETURN
```

Figure 3.9 Generating A BVN Random Variable.

Lines 602-606 generate the target location. Lines 602-604 compute both components of a BVN(0, 0, 1, 1, 0) random variable using two independent uniform (0,1) random numbers generated by the M100's RND(1) function. Equations 3.4, and 3.5, as described in [Ref. 4:page 953], form the basis for lines 602 and 604. Equations 3.4 generate two independent normal(0,1) random variables, X_1' and X_2'.

$$X_1' = (-2\ln U_1)^{.5} \cos(2\pi U_2) \qquad \text{(eqn 3.4)}$$
$$X_2' = (-2\ln U_1)^{.5} \sin(2\pi U_2)$$

Equations 3.5 convert these two independent normal random variables into the components of a BVN(0, 0, 1, 1, ρ) distribution, X_1 and X_2.

$$X_1 = X_1' \qquad \text{(eqn 3.5)}$$
$$X_2 = \rho X_1' + (1-\rho^2)^{.5} X_2'$$

Line 606 scales the components of the BVN(0, 0, 1, 1, $\rho_{x,y}$) by S1 and S2 to produce the components of the BVN(0, 0, σ^2_x, σ^2_y, $\rho_{x,y}$) target location distribution and then returns to the calling program.

Lines 612-616 generate a sensor location using the same algorithm that was used to generate the target location. However, in addition to being scaled by σ_u and σ_v, sensor locations must also be displaced by μ_u and μ_v.

6. **Entering The Number Of Repetitions, Figure 3.10**

```
900 C!S:INPUT"Enter number of repetitions for Monte   Carlo Simulation:";NR
905 RETURN
910 INPUT"Hit ENTER to Continue";Z1:RETURN
```

Figure 3.10 Entering The Number Of Repetitions.

Line 900 prompts the operator to interactively specify the number of repetitions, NR, for Monte Carlo simulations.

Line 910 is a subroutine which stops the program while the operator views output to the screen.

7. **Output Section, Figure 3.11**

All printing is to the screen of the M100. Lines 951-952 play a tune to notify the operator that the calculation is finished. Line 953 also prints the time required to do the calculation and branches around the selection of α for the confidence interval if a numerical approximation was used. Lines 954-959 prompt the operator to select .1, .05, or .01 as α = AL, the probability that the true probability of detection is not in the confidence interval. Once AL is selected, it is reassigned the value of the standard normal at $Z_{1-\alpha/2}$. Line 960 prints the point estimate of P_d. Line 962 prints a short reminder that there is no confidence interval with numerical approximations. Line 966

```
950 'Print output
951 SOUND1567,10:SOUND1244,10:SOUND1046,10:SOUND783,20
952 SOUND1046,10:SOUND783,40
953 CLS:PRINT"":PRINT"Calculation Time (HH/MM/SS) = ";TIME$:IFF3=2THEN960
954 PRINT"Select Alpha for Confidence interval:"
955 INPUT" Choices = .1, .05, .01:";AL
956 IFAL=.1THENAL=1.645:GOTO960
957 IFAL=.05THENAL=1.96:GOTO960
958 IFAL=.01THENAL=2.575:GOTO960
959 GOTO954
960 PRINT"*** Estimate of P(Detection) = ";:PRINTUSING" #.#####";PD
961 IFF3=1THEN965
962 PRINT"No Confidence Interval For Numerical    Approximations"
963 GOTO970
965 PRINT"Confidence Interval: (";
966 TE=AL*SQR(PD*(1-PD)/NR):LL=PD-TE:UL=PD+TE:IFUL>1THENUL=1
967 IFLL<0THENLL=0
968 PRINTUSING"##.#####";LL;UL;:PRINT")":GOSUB910
970 'Confetti Approximation
972 PRINT"":INPUT"Confetti approximation? 0=No, 1=Yes:";Z9:IFZ9=0THENRETURN
974 CLS:INPUT"Enter TOTAL lethal area for ALL sensors   in the pattern:";NA
976 TE=NA/(6.283185*S1*S2):TE=1-(1+SQR(2*TE))*EXP(-SQR(2*TE))
977 PRINT"**Confetti Approximation = ";TE:GOSUB910:RETURN
```

Figure 3.11 Output Section.

calculates the confidence interval according to Equation 3.3. Lines 966-967 ensure that the confidence limits are between zero and one. Lines 965 and 968 print the confidence interval.

Lines 970-977 approximate P_d with the "confetti approximation" described in [Ref. 5:pp. 14-16]. This approximates P_d by distributing the total lethal area of the entire group of sensors over an ellipse as described in Reference 5. This approximation is calculated by Equation 3.6 where ζ = (Total Lethal Area)/$(2\pi\sigma_x\sigma_y)$.

$$P_d = 1 - (1 + (2\zeta)^{.5})\, e^{-(2\zeta)^{.5}} \qquad \text{(eqn 3.6)}$$

Line 972 prompts the operator to indicate whether a confetti approximation is desired and ends the output routine if it is not. Line 974 prompts the operator to enter the total lethal area for all sensors combined. Line 976 computes ζ and then P_d. Line 977 prints P_d and ends the output subroutine.

8. **Numerical Integration, Figure 3.12**

This subroutine is a tailored version of the Romberg integration subroutine documented in Chapter 8. It integrates the target location distribution subject to circular limits of integration.

```
1200 'Numerical Integration Subroutine
1201 D1=6.2831853*S1*S2*RF
1202 TL=.001
1220 CLS:PRINT"":PRINT"      !!Calculating An Integral!!":PRINT""
1230 N=2:GOSUB1293:DY=(YU-YL)/2
1240 FORJ9=1TO6:DY=DY/2:N=N*2
1242 Y=YU:GOSUB1296:GOSUB1280:A2(J9,1)=TS*DX
1245 Y=YL:GOSUB1296:GOSUB1280:A2(J9,1)=A2(J9,1)+TS*DX
1250 FORJ8=2TON:Y=Y+DY:GOSUB1296:GOSUB1280
1251 A2(J9,1)=A2(J9,1)+2*TS*DX:NEXTJ8
1252 A2(J9,1)=A2(J9,1)*DY/2
1255 IFJ9=1THENNEXTJ9
1260 FORJ8=1TOJ9-1
1262 A2(J9,J8+1)=A2(J9,J8)+((A2(J9,J8)-A2(J9-1,J8))/(4^J8-1)):NEXTJ8
1263 T1=A2(J9,J9)-A2(J9,J9-1):IFSGN(T1)*T1-TL>0THENNEXTJ9ELSE1266
1264 PRINT"Tolerance of";TL;"not met after five      extrapolations"
1266 IN=A2(J9,J9):RETURN
1275 FORJ7=1TO6:FORJ6=1TOJ7:PRINTUSING"##.###";A2(J7,J6);
1276 NEXTJ6:PRINT"":NEXTJ7:INPUTZ9:RETURN
1280 REM Trapezoidal Rule Sum
1281 X=XU:GOSUB1286:TS=F:X=XL:GOSUB1286:TS=TS+F
1282 FORJ5=2TON-1:X=X+DX:GOSUB1286:TS=TS+F:NEXTJ5:RETURN
1285 'F(X,Y) to be integrated:
1286 F=X^2/V1-2*RH*X*Y/S1/S2+Y^2/V2
1287 F=(EXP(-F/2/RF^2))/D1:RETURN
1290 'Limits of Integration:
1293 YU=X(J2,2)+H:YL=X(J2,2)-H:RETURN
1296 T3=SQR(H^2-(Y-X(J2,2))^2):XU=X(J2,1)+T3:XL=X(J2,1)-T3:DX=(XU-XL)/N
1297 RETURN
```

Figure 3.12 Numerical Integration Subroutine.

9. Changing The Detection Function, Figure 3.13

```
1300 PRINT" -Detection Fn (DF) in terms of XT, YT,"
1302 PRINT"  and Parameters XS, YS, and X(J2,6),....,X(J2,5+NP):"
1304 PRINT"  ** DF = ";DF$
1306 PRINT"Hit ENTER For No Change or Enter New...":INPUT" DF = ";DF$
1307 RETURN
```

Figure 3.13 Changing The Detection Function.

This section contains a subroutine which permits the operator to interactively change the detection function, lines 1300-1306., These changes would be made if the operator wanted to use a probabilistic detection function other than a Carlton function. In both equations XT and YT represent the two components of the target

location. XS and YS represent the location of S_{J2} in the same coordinate system as XT and YT. An example of a probabilistic detection function that is not Carlton might be an exponential function as specified in Equation 3.7.

$$P_d = \lambda e^{-\lambda r} \quad \text{where } r = \text{distance from sensor to target.} \qquad \text{(eqn 3.7)}$$

The entry to be made when prompted by line 1306 would be

X(J2,6)*EXP(-X(J2,6)*SQR((XT-XS)^2+(YT-YS)^2))

where λ for sensor J2 is X(J2,6). In general, X(J2,6),...,X(J2,5+NP) represent NP other parameters of the selected detection function in addition to the five parameters of the BVN sensor location distribution. The formula for the current detection function is displayed by line 1304. The operator may either change it as desired or hit ENTER to leave the current formula unchanged.

10. **Formula Tokenization Section, Figure 3.14**

```
1400 'Tokenize DF
1410 B$="DF="+DF$+CHR$(0)
1450 'Tokenize/execute B$
1451 B0=VARPTR(B$):B1=PEEK(B0+1)+256*PEEK(B0+2):CALL1606,0,B1
1455 CALL2499,0,63105:RETURN
```

Figure 3.14 Formula Tokenization Section.

The right hand side of the detection function equation is stored as a string variable, DFS. This section converts DFS into an executeable BASIC assignment statement and executes that statement. A detailed explanation of this subroutine is found in the Formula Tokenization Section in Chapter 2.

F. SAMPLE PROBLEMS

Examples 1-6 which follow have the following characteristics in common.

- They use a target location distribution that is BVN(0, 0, 25^2, 25^2, 0), except for Example 6 in which $\rho_{x,y} = .5$.
- All measurements are in kilometers and are in the coordinate system of the target location distribution.
- All confidence intervals are calculated with $\alpha = .05$.

- All Monte Carlo simulations were done with 5,000 repetitions.
- All sensors, deterministic or probabilistic, have a lethal radius of twenty and therefore a lethal area of 400π.

1. **Example 1**

This example is of a single cookie-cutter sensor located at (0,0) with a lethal radius of 20. The input file and diagram of sensor coverage is at Figure 3.15. The closed form solution may be calculated using Equation 3.8.

$$P_d = 1 - e^{-r^2/2\sigma^2} = 1 - e^{-20^2/(2*25^2)} = .27385 \qquad \text{(eqn 3.8)}$$

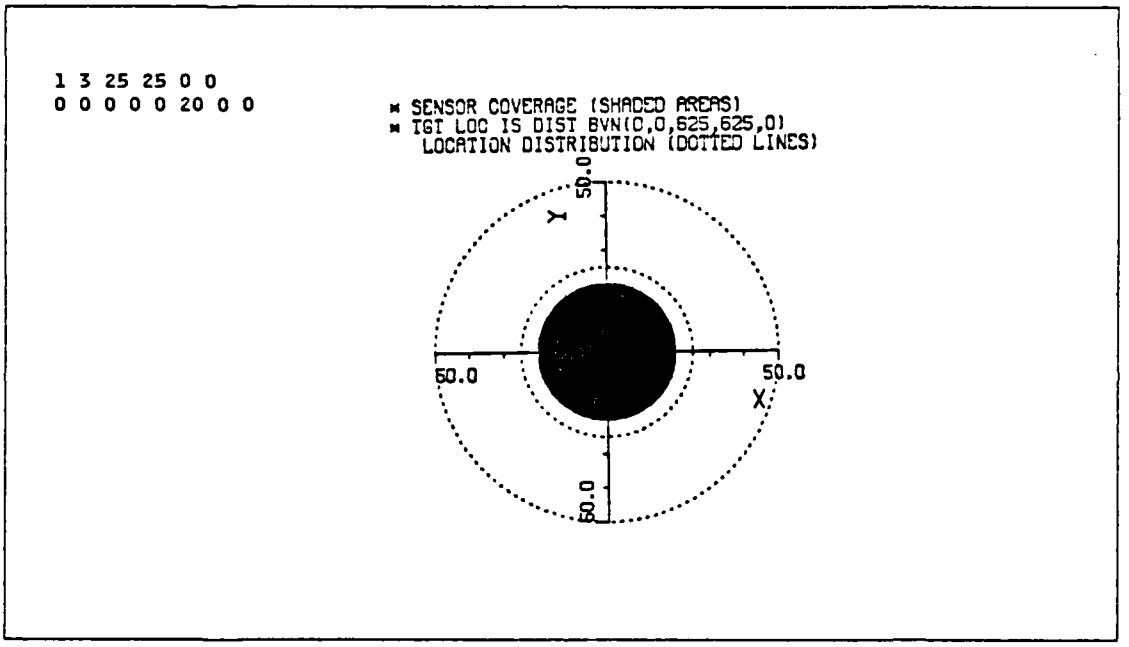


Figure 3.15 Input File And Sensor Coverage Diagram For Example 1.

The probabilities of detection computed by the various computational techniques and calculation times were as follows.

- Closed form: .27385
- Numerical integration: .27252; 3 minutes, 24 seconds
- Monte Carlo simulation: .27640 ± .0124; 4 hours, 22 minutes.

2. **Example 2**

This example is of a single Carlton sensor located at (0,0) with a lethal area equivalent to the cookie-cutter sensor in Example 1; i.e., $b = 20/(2^{.5}) = 14.14214$. The input file and diagram of sensor coverage is shown in [Ref. 5:page 5]. The closed form solution is at Equation 3.9. Because the sensor is located at the origin and $\sigma_x = \sigma_y$, Equation 3.9 simplifies to Equation 3.10 for this example.

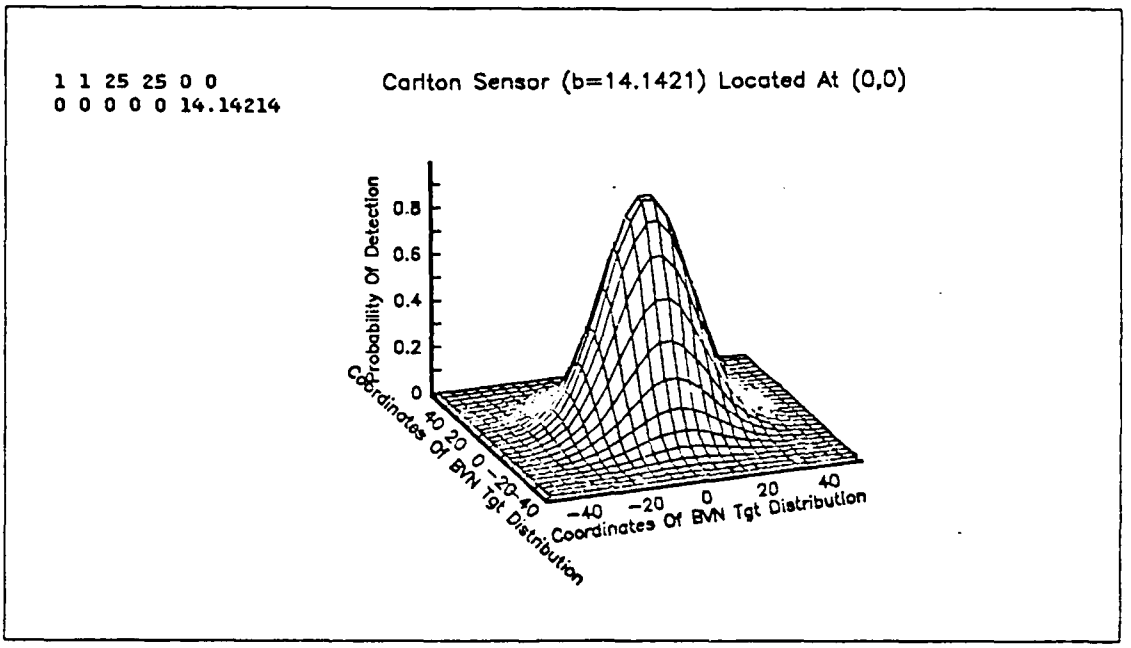


Figure 3.16 Input File And Sensor Coverage Diagram For Example 2.

The probabilities of detection computed by the various computational techniques and calculation times were as follows.

- Closed form: .242424
- Numerical Integration: None
- Monte Carlo simulation: .2452 ± .0119; 3 hours, 44 minutes.

$$P_d = \gamma\, e^{-.5(\delta_u + \delta_v)} \qquad \text{(eqn 3.9)}$$

where $\gamma = b^2/[(b^2+\sigma^2_x)(b^2+\sigma^2_y)]^{.5}$,

$\delta_u = \mu^2_u/(b^2 + \sigma^2_u)$, and $\delta_v = \mu^2_v/(b^2+\sigma^2_v)$.

$$P_d = b^2/(b^2 + \sigma^2) = 14.142^2/(14.142^2 + 25^2) = .242424 \qquad \text{(eqn 3.10)}$$

3. **Example 3**

This example is for a single cookie-cutter sensor *offset at (10,10)* with a lethal radius of 20. The input file and diagram of sensor coverage is shown in Figure 3.17. There is no closed form solution for offset cookie-cutter sensors.

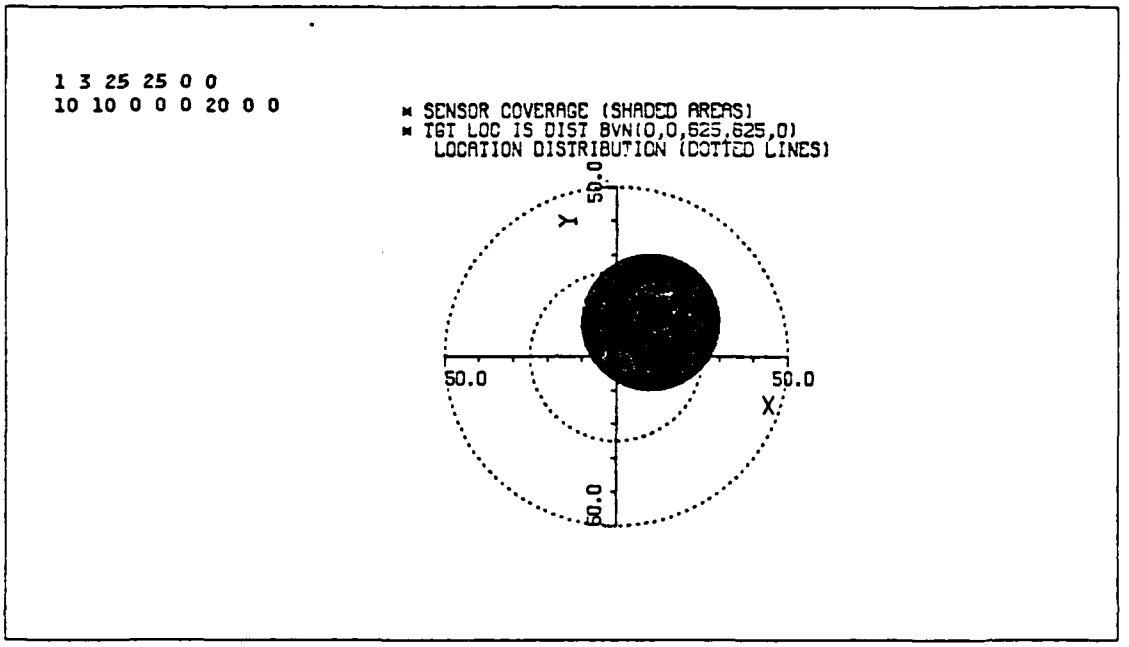


Figure 3.17 Input File And Sensor Coverage Diagram For Example 3.

The probabilities of detection computed by the various computational techniques and calculation times were as follows.

- Closed form: None.
- Numerical integration: .2400; 2 minutes, 51 seconds.
- Monte Carlo simulation: .24160 ± .01187; 2 hours, 57 minutes.

4. **Example 4**

This example is for a single Carlton sensor offset at (10,10) with a lethal area equivalent to the cookie-cutter sensor in Example 3, i.e. $b = 20/(2^{.5}) = 14.14214$ The input file and diagram of sensor coverage is shown in Figure 3.18. The equation for the closed form solution is shown in Equation 3.9.

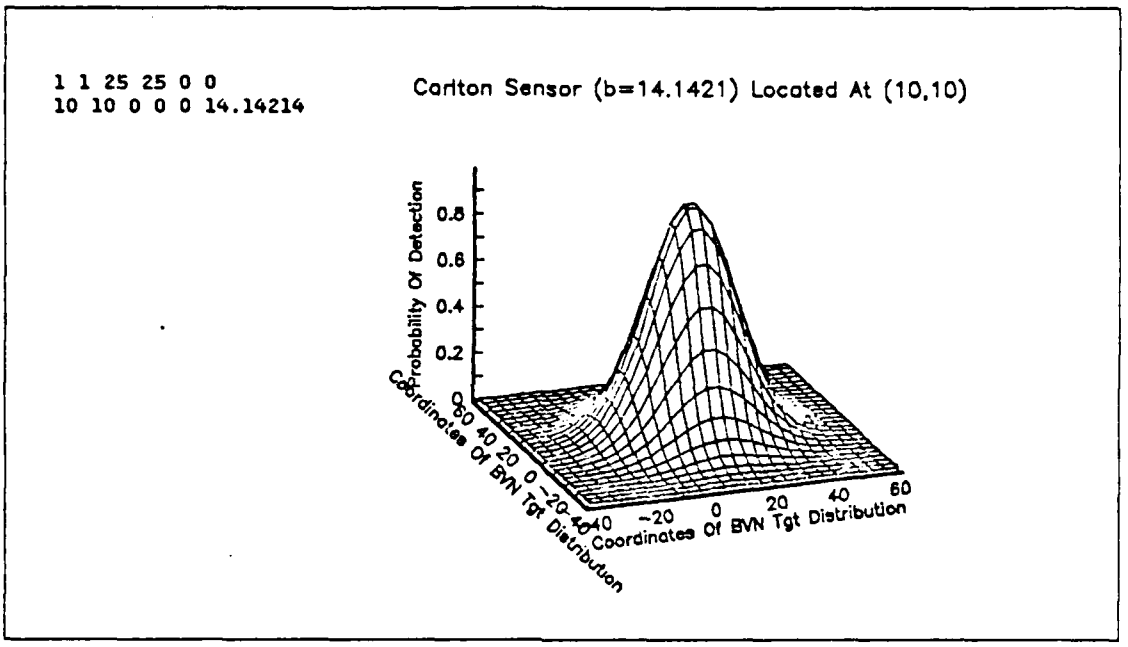


Figure 3.18 Input File And Sensor Coverage Diagram For Example 4.

The probabilities of detection computed by the various computational techniques and calculation times were as follows.

- Closed form: .21475
- Numerical integration: None
- Monte Carlo simulation: .2138 ± .0114; 3 hours, 15 minutes.

5. **Example 5**

This example is for a single convergence zone sensor located (0,0) with a lethal area equivalent to the sensors in Examples 1-4. The sensor detects all targets at ranges less than four, detects no targets between four and 30, detects all targets between 30 and 35.83, and detects no targets beyond 35.83. The input file and sensor coverage diagram is shown in Figure 3.19. The closed form solution is found by using Equation 3.8 to calculate P_d in circles of radii 4, 30, and 35.83. Then $P_d = P_d(4) - P_d(30) + P_d(35.83)$.

The probabilities of detection computed by the various computational techniques and calculation times were as follows.

- Closed form: .141463
- Numerical integration: .14128
- Monte Carlo simulation: .1416 ± .0097; 2 hours, 59 minutes.

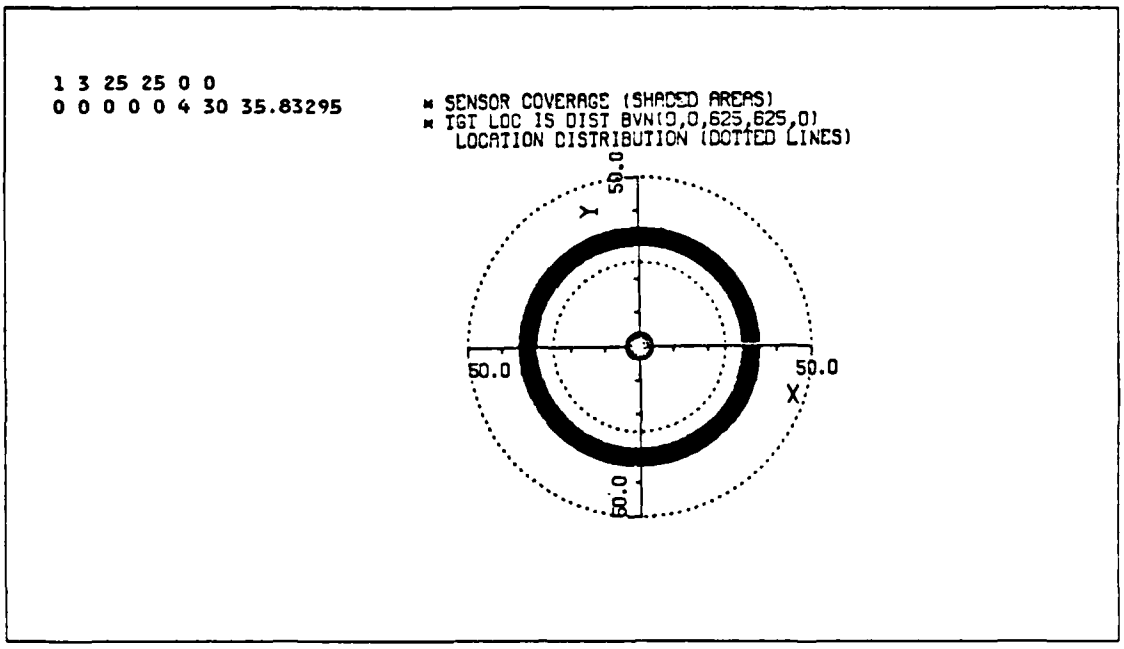


Figure 3.19 Input File And Sensor Coverage Diagram For Example 5.

6. **Example 6**

This example is for four convergence zone sensors with mean locations at (10,10), (-10,-10), (10.-10), and (-10,10). These sensors have convergence zones equal to that of the the sensor in Example 5. Sensor locations are distributed BVN with μ_u and μ_v as indicated above and $\sigma_u = \sigma_v = 3$, and $\rho_{u,v} = .7$. The input file is at Figure 3.20. There is no closed form solution for this example.

```
4 3 25 25 .5 1
10 10 3 3 .7 4 30 35.83295
-10 -10 3 3 .7
10 -10 3 3 .7
-10 10 3 3 .7
```

Figure 3.20 Input File For Example 6.

The probabilities of detection computed by the various computational techniques and calculation times were as follows.

- Closed form: None
- Numerical integration: None

- Monte Carlo simulation: .4570 ± .01381; 2 hours, 53 minutes.

IV. KALMAN FILTER PROGRAM

A. GENERAL

This program successively updates an estimate of the state of a system, μ, based upon a series of measurements, ζ. Vectors μ and ζ need not have the same dimensions. The program makes provision for changes in the system state between measurements in accordance with a linear system model. Covariances between elements of μ and between elements of ζ are also accounted for by the program.

The purpose of this chapter is to describe an implementation of a Kalman filter on the M100. A fuller explanation of the mathematical background behind Kalman filters may be found in References 6 and 7. The notation in the following program is similar to the notation used in Reference 6 to facilitate comparison of the mathematical theory and computer implementation.

The state of the system, X, is assumed to be a multivariate normal random variable, $X \sim N(\mu,\Sigma)$, with system noise, $W \sim (\mu_W,Q)$, and measurement noise, $V \sim (\mu_V,R)$. φ is the matrix which models the linear change in X between measurements. H is the matrix which shows the linear relationship between the measurement and the system state, i.e. how the measurement depends on the system state.

The Kalman filter recursively updates μ by repeating two steps, measurement and movement. The measurement step calculates the Kalman gain, enters a new measurement, and updates μ and Σ based on that measurement. The movement step updates μ and Σ based upon the system model.

B. EXPLANATION OF VARIABLES

- B1 is the intermediate matrix for inversion of C2.
- C1(MD,MD) and C2(MD,MD) are the matrices which hold intermediate results of matrix calculations.
- H is the matrix showing the relationship between measurements and the system state.
- I1, I2, and I3 are loop counters.
- K(NX,NZ) is the Kalman gain matrix.
- MD is the maximum of NX and NZ.
- MU(NX) is the current estimate of the system state.
- MV(NZ) is the mean of measurement noise.
- MW(NX) is the mean of system noise.

- NX is the number of elements in the system state vector.
- NZ is the number of elements in the measurement vector.
- PH(NX,NX) is the matrix, φ, modeling the linear change in the system state between measurements.
- Q(NX,NX) is the covariance matrix of system noise.
- R(NZ,NZ) is the covariance matrix of measurement noise.
- SG(NX,NX) is the covariance matrix of the system state.
- Z(NZ) is the vector holding the current measurement, ζ.

C. INPUT

The operator must create a RAM file, KALIN.DO, which contains the program input. KALIN.DO must contain the following parameters and matrices in order.

- NX and NZ, the number of elements in μ and ζ respectively.
- Matrix PH(NX,NX), the system model matrix, φ.
- Vector MW(NX), the mean of system noise, μ_W.
- Matrix Q(NX,NX), the covariance matrix of system noise.
- Matrix H(NZ,NX), the matrix showing the relationship between μ and ζ.
- Vector MV(NZ), the mean of measurement noise, μ_V.
- Matrix R(NZ,NZ), the covariance matrix of measurement noise.
- Vector MU(NX), the initial estimate of the system state.
- Matrix SG(NX,NX), the initial estimate of the covariance matrix of the system state, Σ.

After the initial estimate of the system state is entered from the input file, KALIN.DO, the program will prompt the operator for measurements and changes to the H matrix to be entered from the keyboard.

D. OUTPUT

All output goes to the screen of the M100. After the measurement step the program displays the updated Kalman gain matrix, the estimated system state, $\mu+$, and the covariance matrix, $\Sigma+$. After the movement step the program displays the estimate of the system state, $\mu-$, and covariance matrix, $\Sigma-$, just prior to the next measurement.

E. EXPLANATION OF PROGRAM

A complete program listing is located at Appendix B.

1. **Initialization/Input, Figure 4.1**

```
100 CLS:PRINT"*****KALMAN FILTER*****":PRINT"   Input Data Being Read"
110 OPEN"KALIN"FORINPUTAS1:ONERRORGOTO9900
120 INPUT#1,NX,NZ:IFNX<NZTHENMD=NZELSEMD=NX
125 DIMPH(NX,NX),MW(NX),Q(NX,NX),H(NZ,NX),MV(NZ),R(NZ,NZ),MU(NX),SG(NX,NX)
126 DIMC1(MD,MD),C2(MD,MD),K(NX,NZ),B1(NZ+1,NZ*2)
130 FORI1=1TONX:FORI2=1TONX:INPUT#1,PH(I1,I2):NEXTI2:NEXTI1
132 FORI1=1TONX:INPUT#1,MW(I1):NEXTI1
134 FORI1=1TONX:FORI2=1TONX:INPUT#1,Q(I1,I2):NEXTI2:NEXTI1
136 FORI1=1TONZ:FORI2=1TONX:INPUT#1,H(I1,I2):NEXTI2:NEXTI1
138 FORI1=1TONZ:INPUT#1,MV(I1):NEXTI1
140 FORI1=1TONZ:FORI2=1TONZ:INPUT#1,R(I1,I2):NEXTI2:NEXTI1
142 FORI1=1TONX:INPUT#1,MU(I1):NEXTI1
144 FORI1=1TONX:FORI2=1TONX:INPUT#1,SG(I1,I2):NEXTI2:NEXTI1
145 CC=0:CLS:PRINT"Initial  SG  As Input Check:":GOSUB532
```

Figure 4.1 Initialization and Input Section.

Line 110 opens the input file, KALIN.DO and branches the program to line 9900 if an error occures. Line 120 enters NX and NZ and calculates MD. Lines 125 and 126 dimension the matrices in the program. Lines 130-144 enter the matrices from KALIN.DO as described in the input section above. Line 145 initializes the measurement counter, CC, and prints the last input matrix, SG, as a check on input accuracy.

2. **Measurement Block, Lines 150-387**

Measurement block matrix equations for updating K, μ, and Σ are listed in Equations 4.1, 4.2, 4.3.

$$K = \Sigma H^t (H \Sigma H^t + R)^{-1} \qquad \text{(eqn 4.1)}$$

$$\mu = \mu + K(Z - \mu_v - H\mu) \qquad \text{(eqn 4.2)}$$

$$\Sigma = (I - KH)\Sigma \qquad \text{(eqn 4.3)}$$

a. Entering A New H Matrix, Figure 4.2

```
150 CLS:PRINT"       *****MEASUREMENT BLOCK*****"
160 PRINT"Current  H ":GOSUB540
162 INPUT"Enter New  H ? 1=Yes, 0=No:";Z9:IFZ9=0THEN170
165 'Enter A New  H
167 FORI1=1TONZ:FORI2=1TONX
168 PRINT"Enter Row";I1;", Column";I2;"Of  H :";
169 INPUTH(I1,I2):NEXTI2:PRINT"":NEXTI1
```

Figure 4.2 Entering a New H Matrix.

Some Kalman filter problems require a different H matrix for each measurement. This section permits such a matrix to be entered. Line 160 prints a header and calls the subroutine which prints the current H matrix. Line 162 asks the operator whether a new H matrix is required and branches the program appropriately. Lines 167-169 prompt the operator to fill the new H matrix row by row.

b. Kalman Gain Calculation, Figure 4.3

```
170 'CALC KALMAN GAIN
171 'MULT  SG  H t, INTO C1
172 FORI1=1TONX:FORI2=1TONZ:C1(I1,I2)=0:FORI3=1TONX
174 C1(I1,I2)=(SG(I1,I3)*H(I2,I3))+C1(I1,I2):NEXTI3:NEXTI2:NEXTI1
180 'MULT  H   SG  H t , INTO C2
182 FORI1=1TONZ:FORI2=1TONZ:C2(I1,I2)=0:FORI3=1TONX
184 C2(I1,I2)=(H(I1,I3)*C1(I3,I2))+C2(I1,I2):NEXTI3:NEXTI2:NEXTI1
200 'ADD  R  INTO C2
202 FORI1=1TONZ:FORI2=1TONZ:C2(I1,I2)=C2(I1,I2)+R(I1,I2)
203 NEXTI2:NEXTI1
210 'INVERT  C2
215 GOSUB9800
220 'MULT  C1  C2  INTO  K
222 FORI1=1TONX:FORI2=1TONZ:K(I1,I2)=0:FORI3=1TONZ
224 K(I1,I2)=(C1(I1,I3)*C2(I3,I2))+K(I1,I2):NEXTI3:NEXTI2:NEXTI1
```

Figure 4.3 Kalman Gain Calculation.

This section updates the Kalman gain matrix, K, in accordance with Equation 4.1. Lines 171-174 multiply Σ by H^t and place the result in matrix C1. Lines 180-184 multiply H by ΣH^t and place the result in matrix C2. Lines 200-203 add R to $(H\Sigma H^t)$ and place the result in C2. Line 215 calls the subroutine which inverts

$(H\Sigma H^t + R)$. Lines 220-224 multiply ΣH^t by $(H\Sigma H^t + R)^{-1}$, producing an updated Kalman gain matrix, K.

c. **Enter A Measurement and Update μ, Figure 4.4**

```
250 '*****UPDATE MU- TO MU+*****
251 'MULT  H  MU-  INTO  C1
252 FORI1=1TONZ:C1(I1,1)=0:FORI3=1TONX
254 C1(I1,1)=(H(I1,I3)*MU(I3))+C1(I1,1):NEXTI3:NEXTI1
260 'ADD  MV +  H  MU-
262 FORI1=1TONZ:C1(I1,1)=C1(I1,1)+MV(I1):NEXTI1
270 'INPUT A NEW MEASUREMENT
272 CC=CC+1:CLS:PRINT"Measurement #";CC;":"
273 FORI1=1TONZ:PRINT"Enter Element";I1;"Of Measurement:";
274 INPUTZ(I1):NEXTI1
280 'SUBTRACT C1 FROM Z, INTO C1
282 FORI1=1TONZ:C1(I1,1)=Z(I1)-C1(I1,1):NEXTI1
290 'MULT  K  C1  INTO  C2
292 FORI1=1TONX:C2(I1,1)=0:FORI3=1TONZ
294 C2(I1,1)=(K(I1,I3)*C1(I3,1))+C2(I1,1):NEXTI3:NEXTI1
300 'ADD  C2 + MU-  TO UPDATE TO  MU+
302 FORI1=1TONX:MU(I1)=C2(I1,1)+MU(I1):NEXTI1
```

Figure 4.4 Enter Measurement And Update The Estimate of μ.

This section enters a new measurement, Z, from the keyboard and updates the estimate of μ in accordance with Equation 4.2. Lines 251-254 multiply H by μ and place the result in C1. Line 262 adds μ_V + Hμ and places the result in C1. Lines 272-274 increment the measurement counter and allow the operator to enter the new measurement, Z. Line 282 subtracts (μ_V + Hμ) from Z, and places the result in C1. Lines 292-294 multiply the Kalman gain, K, by (Z - μ_V - Hμ), and place the result in C2. Line 302 adds μ to K(Z - μ_V - Hμ), producing the revised estimate of μ.

d. **Updating Σ, Figure 4.5**

This section updates the estimate of Σ using Equation 4.3. Lines 322-326 multiply the Kalman gain, K, by H, subtract the result from the identity matrix, I, and place the result in C1. Lines 352-354 multiply (I - KH) by Σ and place the result into C2. This result, the updated Σ, is then copied into SG by line 362.

e. **Printing The Updated K, μ, And Σ, Figure 4.6**

Lines 375-377 print a header and call the printing subroutine for the updated Kalman gain. Lines 380-382 print a header and call the printing subroutine for the updated estimate of the system state, μ. Lines 385-387 print a header and call

```
320 'MULT  K  H  & SUBTR FROM  I , PUT IN C1
322 FORI1=1TONX:FORI2=1TONX:C1(I1,I2)=0:FORI3=1TONZ
324 C1(I1,I2)=(K(I1,I3)*H(I3,I2))+C1(I1,I2):NEXTI3:C1(I1,I2)=-C1(I1,I2)
326 NEXTI2:NEXTI1
328 FORI1=1TONX:C1(I1,I1)=1+C1(I1,I1):NEXTI1
350 'MULT LAST RESULT BY  SG , INTO C2
352 FORI1=1TONX:FORI2=1TONX:C2(I1,I2)=0:FORI3=1TONX
354 C2(I1,I2)=(C1(I1,I3)*SG(I3,I2))+C2(I1,I2):NEXTI3:NEXTI2:NEXTI1
360 'PUT C2 INTO  SG
362 FORI1=1TONX:FORI2=1TONX:SG(I1,I2)=C2(I1,I2):NEXTI2:NEXTI1
```

Figure 4.5 Updating The System Covariance Matrix, Σ.

```
375 CLS:PRINT"Kalman Gain, K(i,j) After"
377 PRINT"Measurement #";CC:GOSUB510
380 CLS:PRINT"Estimate Of System State, MU(i)+ After"
382 PRINT"Measurement #";CC:GOSUB520
385 CLS:PRINT"Estimate Of Covar, SG(i,j)+ After"
387 PRINT"Measurement #";CC:GOSUB530
```

Figure 4.6 Printing The Updated Kalman Gain, μ, And Σ.

the printing subroutine for the updated estimate of covariance matrix of the system state, Σ.

3. **Movement Block, Lines 400-490**

Movement block matrix equations for updating μ and Σ are listed in Equations 4.4, and 4.5.

$$\mu = \varphi\mu + \mu_W \qquad \text{(eqn 4.4)}$$

$$\Sigma = \varphi\Sigma\varphi^t + Q \qquad \text{(eqn 4.5)}$$

***a.* Updating The Estimate of μ, Figure 4.7**

Lines 422-424 multiply φ by μ and place the result in C1. Line 432 adds μ_W to $\varphi\mu$, producing the updated estimate of μ just before measurement CC + 1.

```
400 CLS:PRINT"*********MOVEMENT BLOCK*******"
410 'Update MU(CC)+ to MU(CC+1)-
420 'MULT  PH  MU , PUT IN C1
422 FORI1=1TONX:C1(I1,1)=0:FORI3=1TONX
424 C1(I1,1)=(PH(I1,I3)*MU(I3))+C1(I1,1):NEXTI3:NEXTI1
430 'ADD C1+ MW , INTO  MU
432 FORI1=1TONX:MU(I1)=C1(I1,1)+MW(I1):NEXTI1
```

Figure 4.7 Updating The Estimate of μ.

b. **Updating Σ, Figure 4.8**

```
440 '**UPDATE  SG **
450 'MULT  PH  SG , INTO C1
452 FORI1=1TONX:FORI2=1TONX:C1(I1,I2)=0:FORI3=1TONX
454 C1(I1,I2)=(PH(I1,I3)*SG(I3,I2))+C1(I1,I2):NEXTI3:NEXTI2:NEXTI1
460 'MULT  C1  PH t, INTO C2
462 FORI1=1TONX:FORI2=1TONX:C2(I1,I2)=0:FORI3=1TONX
464 C2(I1,I2)=(C1(I1,I3)*PH(I2,I3))+C2(I1,I2):NEXTI3:NEXTI2:NEXTI1
470 'ADD  C2 + Q = SG
472 FORI1=1TONX:FORI2=1TONX:SG(I1,I2)=C2(I1,I2)+Q(I1,I2)
473 NEXTI2:NEXTI1
```

Figure 4.8 Updating Σ.

Lines 452-454 multiply φ by Σ and place the result in C1. Lines 462-464 multiply $\varphi\Sigma$ by φ^t and place the result in C2. Lines 472-473 add Q to $\varphi\Sigma\varphi^t$, producing the updated estimate for Σ just before measurement CC + 1.

c. **Printing The Updated μ And Σ, Figure 4.9**

```
480 PRINT"Estimate Of System State, MU(I)-"
482 PRINT"Before Measurement #";CC+1:GOSUB520
485 CLS:PRINT"Estimate Of Covar, SG(I,J)- Before"
487 PRINT"Measurement #";CC+1:GOSUB530
490 GOTO160
```

Figure 4.9 Printing The Updated μ And Σ.

Lines 480-482 print a header and call the printing subroutine for the updated estimate of the system state, μ. Lines 485-487 print a header and call the printing subroutine for the updated estimate of the system state covariance matrix, Σ. Line 490 branches the program back to the beginning of the measurement block.

4. **Printing Subroutines, Figure 4.10**

Lines 500-554 contain subroutines which print the Kalman gain matrix, μ, Σ, the H matrix, or the C2 matrix.

```
500 'PRINTING SUBROUTINES
510 'PRINT KALMAN GAIN, K
512 FORI1=1TONX:FORI2=1TONZ:PRINTUSING"######.##";K(I1,I2);:NEXTI2
514 PRINT"":NEXTI1:INPUT"Hit ENTER To Continue:";Z9:RETURN
520 'PRINT  MU
522 FORI1=1TONX:PRINTUSING"######.##";MU(I1);:NEXTI1:PRINT""
524 INPUT"Hit ENTER To Continue:";Z9:RETURN
530 'PRINT COVAR MATRIX,  SG
532 FORI1=1TONX:FORI2=1TONX:PRINTUSING"#####.##";SG(I1,I2);:NEXTI2
534 PRINT"":NEXTI1:INPUT"Hit ENTER To Continue:";Z9:RETURN
540 'PRINT  H
542 FORI1=1TONZ:FORI2=1TONX:PRINTUSING"######.##";H(I1,I2);:NEXTI2
544 PRINT"":NEXTI1:RETURN
550 PRINT" C2 MATRIX:"
552 FORI1=1TOA:FORI2=1TOB:PRINTUSING"######.##";C2(I1,I2);:NEXTI2
554 PRINT"":NEXTI1:INPUT"Hit ENTER To Continue:";Z9:RETURN
```

Figure 4.10 Printing Subroutines.

5. **Inversion Subroutine For C2, Figure 4.11**

This subroutine is essentially the same as the matrix inversion subroutine in the matrix algebra program discussed in Appendix E. It has been abbreviated to invert only matrix C2 instead of inverting several matrices of various dimensions as in Appendix E. This subroutine also does not calculate the determinant of C2 to test for invertability. If C2 is not invertable, an division by zero error will occur and the program will branch to the error identification section. A detailed explanation of how the matrix inversion subroutine functions is located in Appendix E.

6. **Error Identification, Figure 4.12**

Line 9900 prints a message indicating that C2 is not invertable. Line 9900 is based upon the assumption that a division by zero error in the inversion subroutine means that C2 is not invertable. Line 9905 prints the error code and line number of other errors. See page 217 of Reference 1 for an explanation of error codes.

```
9800 'INVERT C2
9815 FORI1=1TONZ:FORI2=1TONZ:B1(I1,I2)=C2(I1,I2):NEXTI2:NEXTI1
9820 FORI1=NZ+1TO2*NZ:FORI2=1TONZ
9822 IFI1=I2+NZTHENB1(I2,I1)=1ELSEB1(I2,I1)=0
9825 NEXTI2:NEXTI1
9830 FORI1=1TONZ
9840 ML=1/B1(I1,I1):FORI3=1TO2*NZ:B1(I1,I3)=B1(I1,I3)*ML:NEXTI3
9842 IFI1=NZTHEN9865
9845 FORI2=I1+1TONZ:IFB1(I2,I1)=0THEN9860
9850 ML=-B1(I2,I1)
9855 FORI3=I1TO2*NZ:B1(I2,I3)=B1(I2,I3)+(ML*B1(I1,I3)):NEXTI3
9860 NEXTI2:NEXTI1
9865 FORI1=NZTO2STEP-1
9870 FORI2=I1-1TO1STEP-1:IFB1(I2,I1)=0THEN9885
9875 ML=-B1(I2,I1)
9880 FORI3=1TO2*NZ:B1(I2,I3)=B1(I2,I3)+(ML*B1(I1,I3)):NEXTI3
9885 NEXTI2:NEXTI1
9890 FORI1=1TONZ:FORI2=1TONZ
9895 C2(I2,I1)=B1(I2,I1+NZ):NEXTI2:NEXTI1
9897 MI=1:RETURN
```

Figure 4.11 Inversion Subroutine For C2.

```
9900 IFERL>9800ANDERR=11THENPRINT"!!!ERROR:  C2  Is Not Invertable!!!":END
9905 PRINT"Error Code";ERR;"In Line";ERL:END
```

Figure 4.12 Error Identification.

V. MODELS OF COMBAT USING LANCHESTER EQUATIONS

A. GENERAL

This program is an example of a time step force attrition simulation using Lanchester equations. The scenario is that there are two sides, refered to hereafter as the attacker and the defender.[5] The battle may be broken into phases if some model parameters change during the course of the battle. Each side has a fixed number of weapon types throughout the battle. The number of attacking and defending weapon types may be different. The following characteristics must be specified for each weapon type on each side and do not change over the course of the battle.

- The number of weapons at the start of the battle.
- The break point, i.e. the fraction of the starting number of weapons which, if reached, would cause the battle to end. For example, if the attacker would withdraw if 50% of a certain weapon type was lost, then the breakpoint for that weapon type would be .5.
- The Lanchester weapon characteristic, i.e. whether it is a square law, linear law, logarithmic law weapon, or a hybrid.

The following characteristics of each weapon type may change from phase to phase of the battle.

- The time required to complete that phase.
- The rate at which replacements arrive for each weapon type.
- Attrition coefficients, i.e. the rate at which a weapon type is attritted by each opposing weapon type in a particular battle phase.

If there are not more than five weapon types per side, the program prints a dynamic display to the M100 screen showing for each weapon type the fraction of starting strength that has survived and the breakpoint. Regardless of the number of weapon types, the program creates an output file which shows the number of survivors at the end of each phase, which weapon reached its breakpoint first, and the number of survivors at the end of the battle.

---

[5] The terms attacker and defender are arbitrary and are used only for notational purposes.

B. REPRESENTING LANCHESTER CHARACTERISTICS OF EACH WEAPON

The program includes provisions for calculating attrition with traditional Lanchester square law and linear law equations or using a Helmbold equation. Traditional Lanchester attrition uses different functions for different types of attrition. The linear law functions (see Equations 5.1) for changes in strength with respect to time are used to model fire that is aimed at a general area in which targets are believed to be located An example of a linear law weapon would be artillery fire without correction by an observer.

$$dx/dt = -axy, \text{ and} \qquad \text{(eqn 5.1)}$$
$$dy/dt = -byx$$

The square law functions (see Equations 5.2) are used for fire that is aimed at a point instead of a general area.

$$dx/dt = -ay, \text{ and} \qquad \text{(eqn 5.2)}$$
$$dy/dt = -bx$$

The logarithmic law functions (see Equations 5.3) are used to model non-combat losses such as disease.

$$dx/dt = -ax, \text{ and} \qquad \text{(eqn 5.3)}$$
$$dy/dt = -by$$

The reasons for using these equations for modeling these types of attrition are explained in [Ref. 9:Chapter 2]. The limitation of using Equations 5.1, 5.2, and 5.3 is that they restrict the model to three discrete weapons types. However, there may be weapons that do not fall cleanly into any of the three weapon types. For example, artillery fire that is corrected by an observer may have a mixture of linear law and square law characteristics. The traditional Lanchester equations also assume that the full fire power of all the weapons on one side can be brought to bear against all the targets on the opposing side. However, in battles where one side vastly outnumbers the other, or when there are significant terrain masking or reaction time effects, this assumption is not valid. To account for these limitations, R. Helmbold proposed a set of modified Lanchester equation in Reference 8. Helmbold's equations and their empirical validation are also discussed in [Ref. 9:pp. 174-181 and Footnote 2.40]. The special cases of the Helmbold equations used in this program are those listed as Equations 5.4.

$$dx/dt = -a(x/y)^{\omega}y = -ax^{\omega}x^{1-\omega} \qquad \text{(eqn 5.4)}$$
$$dy/dt = -b(y/x)^{\omega}x = -by^{\omega}x^{1-\omega}$$

They include an additional factor, $(x/y)^{\omega}$ or $(y/x)^{\omega}$. If $\omega = 1$, the result is the logarithmic law equation. If $\omega = 0$, the result is the square law equation. If $\omega = .5$, the result is an equation which behaves quite like the linear law equation. ω may be set at any value between zero and one to characterize weapons which do not fall neatly into one of the discrete weapon types.

In the traditional Lanchester linear law equation dx/dt varies proportionately to the number of X survivors and the number of Y survivors. In the Helmbold equation when $\omega = .5$, dx/dt varies proportionately to the square root of the number of X and Y survivors. If dx/dt were equal for the Lanchester linear law and the Helmbold equation with $\omega = .5$, then $a_L xy = a_H(xy)^{.5}$ where a_L and a_H are the Lanchester and Helmbold attrition coefficients respectively. Thus, $a_H = a_L(xy)^{.5}$. Therefore, there is no single value of a_H that will be equivalent to a value of a_L throughout an entire simulation because x and y are changing. However, the general shapes of the functions $k_1 = xy$ and $k_2 = (xy)^{.5}$ are similar enough that they cause attrition to behave about the same in both circumstances. A plot of contour lines of dx/dt = 1, 2, and 3 for linear law and comparable Helmbold equations is shown in Figure 5.1 where $a_H = .02$ and $a_L = .0002$. The contour $dx/dt = 2$ is the same for both formulations.[6] For $dx/dt < 2$ the Helmbold equations produce smaller attrition rates than do the traditional Lanchester equations. For $dx/dt > 2$ the Helmbold equations produce larger attrition rates than do the traditional lanchester equations.

The program includes BASIC code for both the traditional Lanchester and Helmbold equations. To use the traditional Lanchester equations, lines 223-224 and 233-234 should be active, and lines 225 and 235 should be commented out or deleted. To use the Helmbold equations, lines 225 and 235 must be active, and lines 223-224 and 233-234 must be commented out or deleted. *The weapon type parameters in the input file must match the equations which are active in the program.*

---

[6]The Helmbold coutours were jittered slightly to avoid being overprinted at $dx/dt = 2$ by the Lanchester contour.

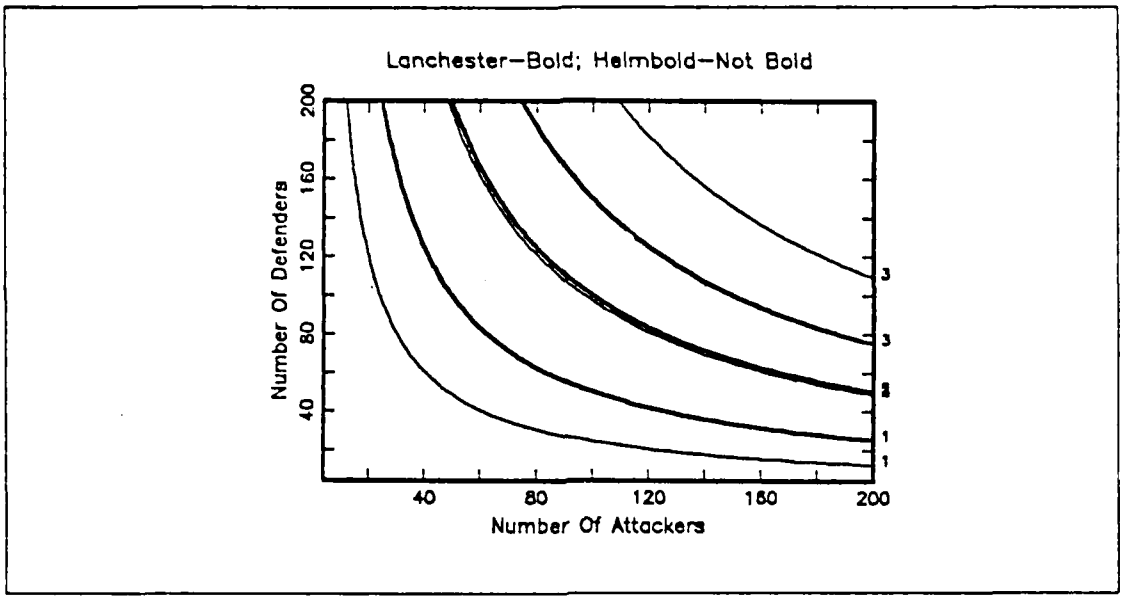


Figure 5.1 dx/dt For Linear Law and Helmbold Equations.

C. EXPLANATION OF VARIABLES

- AA(NA,ND) is a matrix of the rates at which attacking weapons by type are attritted by each type of defending weapon in a particular phase of the battle.
- AB(2,NA) is the matrix for the breakpoints of each attacking weapon type. Elements of the first row, AB(1,i), are the breakpoints expressed as fractions of the starting total, i.e $0 \leq AB(1,i) \leq 1$ for i= 1,2,3...NA. Elements of the second row, AB(2,i), are breakpoints expressed as numbers of weapons of type i, i.e. $0 \leq AB(2,i) \leq SD(i)$ for i= 1,2,3...NA.
- AT(NA) is a vector of tuning parameters that specify the Lanchester characteristics of each attacking weapon type. If line 235 is active and lines 233 and 234 are commented out, then attrition is calculated using the Helmbold equation, see equation 5.4. AT(i) is ω and equals 0 for an aimed fire/square law weapon, .5 for a weapon similar to a Lanchester area fire/linear law weapon, and 1 for a source of non-combat/logarithmic law casualties to the defender. If lines 233 and 234 are active and line 235 is commented out, then attrition is calculated using standard linear law and square law Lanchester equations, see equations 5.1 and 5.2. AT(i) equals 1 for linear law weapons and 2 for square law weapons.

- BB(ND,NA) is a matrix of the rates at which one item of each attacking weapon type attrits each defending weapon type in a particular phase of the battle.
- DB(2,ND) is the matrix for the breakpoints of each defending weapon type. Elements of the first row, DB(1,j), are the breakpoints expressed as fractions of the starting total, i.e $0 \leq DB(1,j) \leq 1$ for j= 1,2,3...ND. Elements of the second row, DB(2,j), are breakpoints expressed as numbers of weapons of type j, i.e. $0 \leq DB(2,j) \leq SD(j)$ for j= 1,2,3...ND.
- DT(ND) is a vector of tuning parameters that specify the Lanchester characteristics of each defending weapon type. If line 225 is active and lines 223 and 224 are commented out, then attrition is calculated using the Helmbold equation, see equation 5.4. DT(j) is ω and equals 0 for an aimed fire/square law weapon, .5 for a weapon similar to a Lanchester area fire/linear law weapon, and 1 for a source of non-combat/logarithmic law casualties to the attacker. If lines 223 and 224 are active and line 225 is commented out, then attrition is calculated using standard linear law and square law Lanchester equations, see Equations 5.1 and 5.2. DT(j) equals 1 for linear law weapons and 2 for square law weapons.
- I1,I2,I3, and I4 are loop counters.
- MD is the maximum of NA and ND.
- NA is the number of attacking weapon types.
- ND is the number of defending weapon types.
- NI is the number of intervals into which a phase of battle is broken.
- NP is the number of phases of battle.
- OA(NA) is the vector of horizontal pixel positions for the current screen representation of the fraction of surviving attackers by weapon type.
- OD(ND) is the vector of horizontal pixel positions for the current screen representation of the fraction of surviving defenders by weapon type.
- QA(2,NA) holds the surviving number of each attacking weapon type. QA(1,i) is the number at the start of a time increment, DT. QA(2,i) is the number after attrition by each defending weapon is subtracted.
- QD(ND) holds the surviving number of each defending weapon type after attrition by each attacking weapon is subtracted.
- SA(NA) is the number of attacking weapons by type at the start of the battle.
- SD(ND) is the number of defending weapons by type at the start of the battle.

- TF is the termination flag. TF = 0 means a breakpoint has not been reached. TF = 1 means a breakpoint has been reached.
- TP is the top pixel position for a rectangle or line drawn on the graphical display.
- TT is the total time that has elapsed in the battle up to the current time increment. When a breakpoint is reached, or when all phases of the battle are completed, TT is time length of the battle reported to LANOUT.DO.

D. INPUT

All input to the program is entered into a RAM file, LANIN.DO. The following parameters must be entered in the following order and do not change between battle phases.

- NP,NA,ND
- QA(NA)
- QD(ND)
- AB(NA)
- DB(ND)
- AT(NA)
- DT(ND)

The following parameters must be entered in the following order for each phase.

- TT, NI
- AR(NA)
- DR(NR)
- AA(NA,ND)
- BB(ND,NA)

An example of an input file is shown in Figure 5.2

The situation to be simulated using the parameters in Figure 5.2 is as follows. The battle has two phases.

1. Parameters Common To Both Phases

The attacker has two weapon types, A_i, i = 1 and 2. The defender has three weapon types, D_j, i = 1, 2, and 3. The attacker starts with 200 A_1's and 100 A_2's. The defender starts with 100 D_1's, 200 D_2's, and 100 D_3's. The battle will end when the first attacking or defending weapon type is attritted to 50% of its initial strength, i.e. the breakpoint for all weapon types is .5. If this simulation is to be run using traditional Lanchester equations, both attacking weapons are square law weapons. D_1 is an linear law weapon, and D_2 and D_3 are square law weapons.

```
2 2 3
200 100
100 200 100
.5 .5
.5 .5 .5
2 2
1 2 2
5 50
1 1
.8 .8 .8
.00012 .014 .016
.00018 .020 .022
.019 .017
.015 .013
.011 .009
10 50
.5 .5
.4 .4 .4
.00018 .021 .024
.00027 .030 .033
.0285 .0255
.0225 .0195
.0175 .00135
```

Figure 5.2 Sample Input File.

2. **Situation For Phase One**

The length of the first phase is five hours. The program will break that period into 50 segments for computational purposes. Replacements arrive for both attacking weapon types at an average rate of one weapon per hour. The replacement rate for all three defending weapon types averages .8 weapons per hour. A_1's are attritted by each D_1 at a rate of .00012 per hour per surviving A1. A_1's are attritted by each D_2 at a rate of .014 per hour. Rates at which the remaining A_i are attritted by each D_j are listed through the end of the next line. D_1's are attritted by each A_1 at a rate of .019 per hour. D_1's are attritted by each A_2 at a rate of .017 per hour. Rates at which the remaining D_j are attritted by each A_i are listed through the end of the next two lines.

3. **Situation For Phase Two**

The length of the second phase is ten hours. The program will break that period into 50 segments for computational purposes. Replacements arrive for both attacking weapon types at the rate of .5 weapons per hour. The replacement rate for all three defending weapon types averages .4 weapons per hour. A_1's are attritted by each D_1 at a rate of .00018 per hour per surviving A1. A_1's are attritted by each D_2 at a rate of .021 per hour. Rates at which the remaining A_i are attritted by each D_j

are listed through the end of the next line. D_1's are attritted by each A_1 at a rate of .0285 per hour. D_1's are attritted by each A_2 at a rate of .0255 per hour. Rates at which the remaining D_j are attritted by each A_i are listed through the end of the next two lines.

E. OUTPUT

The program produces two types of output:

- An output file, LANOUT.DO, which lists the status of each weapon type at the end of each phase and at the end of the battle and which weapon types have gone below their breakpoints.
- A dynamic graphical display to the screen of the M100 which shows the fraction of the starting strength of each weapon type which has survived until that time interval in the simulation.

The graphical display consists of a rectangle on the M100 screen for each attacking and defending weapon type. Each rectangle is 100 pixels wide and five pixels high. Each pixel in the horizontal direction represents one percent of the starting strength of the weapon type represented by that particular rectangle. In each rectangle is a vertical line showing the breakpoint for that weapon type as a fraction of starting strength. As the simulation progresses, another vertical line in each rectangle is updated showing the fraction of survivors for that weapon type. The rectangles are arranged in two columns, one for attacking and one for defending weapon types. Weapon type numbers are printed to the left of the rectangles. If the replacement rate drives the number of survivors over the starting strength for some weapon type, the vertical line indicating the fraction of survivors will stay at the 100% level and an asterisk will be printed to the left of the corresponding rectangle. In addition to the rectangles there is a printed line at the bottom of the screen which tells the operator on what phase and time interval the simulation is currently working.

F. EXPLANATION OF PROGRAM COMPONENTS

A complete listing of this program is at Appendix C.

1. Initialization Section, Figure 5.3

Line 120 sets the number of files to two and opens the input file, LANIN.DO. Line 121 opens the output file, LANOUT.DO and enters the number of phases in the battle, NP, and the number of attacking and defending weapon types, NA and ND. Line 122 defines MD as the maximum of NA and ND. If both sides have five or fewer weapons types, line 124 sets SF = 1, indicating that the screen of the M100 is big enough to handle the graphical display generated during the simulation. If either side

```
100 'LANCHESTER TIME STEP MODEL
120 MAXFILES=2:OPEN"LANIN"FORINPUTAS1
121 OPEN"LANOUT"FOROUTPUTAS2:INPUT#1,NP,NA,ND
122 IFNA>NDTHENMD=NAELSEMD=ND
124 IFMD<6THENCLS:SF=1
130 DIMAA(NA,ND),BB(ND,NA),AT(NA),DT(ND),AR(NA),DR(ND)
131 DIMQA(2,NA),QD(ND),AB(2,NA),DB(2,ND),SA(NA),SD(ND),OA(NA),OD(ND)
```

Figure 5.3 Initialization Section.

has more than five weapon types, the graphical display will not be generated. Lines 130-131 dimension the matrices used in the program.

2. **Entering Common Parameters, Figure 5.4**

```
132 'Enter Initial Quantities of Wpns, Break Points And Wpn Types.
134 FORI2=1TONA:INPUT#1,QA(2,I2):SA(I2)=QA(2,I2):OA(I2)=127:NEXTI2
135 FORI2=1TOND:INPUT#1,QD(I2):SD(I2)=QD(I2):OD(I2)=238:NEXTI2
136 FORI2=1TONA:INPUT#1,AB(1,I2):AB(2,I2)=AB(1,I2)*QA(2,I2):NEXTI2
137 FORI2=1TOND:INPUT#1,DB(1,I2):DB(2,I2)=DB(1,I2)*QD(I2):NEXTI2
138 FORI2=1TONA:INPUT#1,AT(I2):NEXTI2:FORI2=1TOND:INPUT#1,DT(I2):NEXTI2
140 TM=0:IFSF=1THENGOSUB600
```

Figure 5.4 Entering Common Parameters.

This section enters the rest of the parameters that do not change from phase to phase of the battle and initializes the variables controlling the graphical display. Line 134 puts the starting quantity of A_i into QA(2,i) and SA(i) and sets OA(i) to 127 which indicates that 100% of A_i are surviving at the start of the simulation. Line 135 performs the same function for D_j that line 134 performed for A_i. Lines 136 and 137 enter the fractional breakpoints for A_i and D_j respectively into AB(1,i) and DB(1,j). Lines 136 and 137 also compute the breakpoints in terms of numbers of surviving weapons, placing them in AB(2,i) and DB(2,j). Line 138 enters the Lanchester characteristic parameters, AT(i) and DT(j), for each weapon type. Line 140 sets TM, which keeps track of the time until the end of the battle, to zero. If the graphical display to the screen is to be used, line 140 also calls subroutine 600 which sets up the output screen.

3. **Initialization For Eacl Phase, Figure 5.5**

```
143 FORI1=1TONP:PRINT#2,"STARTING PHASE";I1
145 'Enter Time Spent In Phase I1 and # of Intervals
146 INPUT#1,TT,NI:DT=TT/NI
150 'Enter Replacement Rates And Attrition Coefficient Matrices
152 FORI2=1TONA:INPUT#1,AR(I2):NEXTI2:FORI2=1TOND:INPUT#1,DR(I2):NEXTI2
154 FORI2=1TONA:FORI3=1TOND:INPUT#1,AA(I2,I3):NEXTI3:NEXTI2
158 FORI2=1TOND:FORI3=1TONA:INPUT#1,BB(I2,I3):NEXTI3:NEXTI2
```

Figure 5.5 Initialization For Each Phase.

Line 143 sets I1, the phase counter, and prints a heading to the output file. Line 146 enters the length of the current phase, TT, and the number of intervals, NI, into which the phase will be broken and computes the length of each interval, DT. The choice of NI is a compromise between two competing objectives: accuracy and time required to complete the simulation. As NI becomes larger, DT becomes smaller and the simulation more closely approximates the continuous, mutual attrition that is the basis of Lanchester equation theory. If NI is small and DT is large, then the simulation tends to discount the attrition which takes place during a phase because the program assumes force levels are constant throughout an interval, DT. However, if NI is to large, the time required to run the simulation increases linearly. The relationship between accuracy and speed is also a function of the attrition coefficients and number of weapon types on each side.

Line 152 enters the replacement rates for each weapon type. Lines 154-156 enter the attrition coefficients for each weapon type.

4. **Attrition Calculations For A Phase, Figure 5.6**

This section breaks a phase into NI intervals of length DT, calculates the attrition during each interval, and tests whether that attrition has caused some weapon type to reach its breakpoint.

Line 202 sets the interval counter, I2, adds the length of the interval to the length of the battle, TM, and prints a message to the screen telling the operator what phase and interval is currently being processed.

Lines 222-227 calculate the attrition to attackers based upon the quantities of each weapon surviving at the start of the interval. QA(2,i) holds the current quantity

```
200 'Fight Phase I1.
202 FORI2=1TONI:TM=TM+DT:PRINT@241,"Phase:";I1;", Increment";I2;"out of";NI
210 'Fight Time Increment DT.
220 'Update number of attackers
222 FORI3=1TONA:QA(1,I3)=QA(2,I3):NEXTI3:FORI3=1TONA:FORI4=1TOND
223 IFDT(I4)=1THENQA(2,I3)=QA(2,I3)-AA(I3,I4)*QD(I4)*QA(2,I3)*DT:GOTO226
224 QA(2,I3)=QA(2,I3)-AA(I3,I4)*QD(I4)*DT
225 'QA(2,I3)=QA(2,I3)-AA(I3,I4)*(QA(2,I3)/QD(I4))^DT(I4)*QD(I4)*dt
226 NEXTI4:QA(2,I3)=QA(2,I3)+AR(I3)*DT:IFSF=1THENGOSUB650
227 NEXTI3
230 'Update number of defenders
232 FORI3=1TOND:FORI4=1TONA
233 IFAT(I4)=1THENQD(I3)=QD(I3)-BB(I3,I4)*QD(I3)*QA(1,I4)*DT:GOTO236
234 QD(I3)=QD(I3)-BB(I3,I4)*QA(1,I4)*DT
235 'QD(I3)=QD(I3)-BB(I3,I4)*(QD(I3)/QA(1,I4))^AT(I4)*QA(1,I4)*dt
236 NEXTI4:QD(I3)=QD(I3)+DR(I3)*DT:IFSF=1THENGOSUB660
237 NEXTI3
240 GOSUB300:NEXTI2
242 IFI1=NPTHENGOSUB350:CLS:PRINT"Output is in file LANOUT.DO.":END
245 PRINT#2,"Status After Phase";I1:GOSUB361:NEXTI1
```

Figure 5.6 Attrition Calculations for a Phase.

of A_i and is therefore decremented by lines 222-227. Attrition to each D_j should, for consistency with the attrition to the A_i's, also be based upon the quantity of A_i's surviving at the *beginning* of the interval. Therefore, the quantity of each A_i surviving at the beginning of the interval is saved by line 222 in QA(1,i) for use during the attrition calculations for D_j. Line 222 also sets the A_i counter, I3, and the D_j counter, I4.

If the simulation is to be done with traditional Lanchester linear law and square law equations, lines 223-224 must be active and line 225 must be commented out or deleted. If the simulation is to be done with a Helmbold equation, lines 223-224 must be commented out or deleted and line 225 must be active. The program displayed in Figure 5.6 has the Helmbold equations commented out. Line 222 calculates the attrition of A_i by D_j based upon a linear law, Equation 5.1. Line 223 calculates attrition based upon the square law, Equation 5.2. Whether the linear law or square law is used is based upon AT(i) and is determined in line 222. If line 225 were active, it would calculate attrition using the Helmbold equation, Equation 5.4. After attrition by each D_j is calculated, line 226 adds the replacements for A_i received during the interval. Line 226 also calls subroutine 650 which updates the graphical display to reflect the attrition to each A_i.

Lines 232-237 calculate the attrition to defenders based upon the quantities of each weapon surviving at the start of the interval. QD(i) holds the current quantity of D_j. Line 232 sets the D_j counter, I3, and the A_i counter, I4. If the simulation is to be done with traditional Lanchester linear law and square law equations, lines 233-234 must be active and line 235 must be commented out or deleted. If the simulation is to be done with a Helmbold equation, lines 233-234 must be commented out or deleted and line 235 must be active. The program displayed in Figure 5.6 has the Helmbold equations commented out. Line 232 calculates the attrition of D_j by A_i based upon a linear law, Equation 5.1. Line 233 calculates attrition based upon the square law, Equation 5.2. Whether the linear law or square law is used is based upon DT(i) and is determined in line 232. If line 235 were active, it would calculate attrition using the Helmbold equation, Equation 5.4. After attrition by each A_i is calculated, line 236 adds the replacements for D_j received during the interval. Line 236 also calls subroutine 660 which updates the graphical display to reflect the attrition to each D_j. Line 240 calls subroutine 300 to check for whether a breakpoint was reached during the interval. If no breakpoint was reached, a new interval is begun.

If all the intervals in the last phase are completed without reaching a breakpoint, line 242 calls subroutine 350 which prints the status at the end of the battle to the output file. If the current phase is not the last phase, then line 245 prints a header to the output file, calls subroutine 361 which prints the status at the end of the current phase to the output file, and starts the next phase.

5. **Breakpoint Subroutine, Figure 5.7**

This section determines whether any weapons have reached their breakpoints, and, if so, prints that information in the output file, and ends the program. Line 320 sets TF = 0, indicating that no breakpoints have been reached. Line 320 then starts a loop which tests whether any A_i have reached their breakpoints. If so, then lines 322-324 print a message to the output file specifying the weapon type, the breakpoint, and the quantity of that weapon type that survived. Lines 335-340 test whether any D_j have reached their breakpoints, and if so, print a message to that effect to the output file. If no breakpoints have been reached, then line 340 returns control to the main program to begin attrition calculations in the next time interval.

The default battle termination criterion is that at least one weapon type must be below its individual breakpoint at the end of a time interval. However, the operator may wish to edit the program before running it, adding more sophisticated termination

```
300 'Check Whether Breakpoint is reached.
320 TF=0:FORI3=1TONA:IFQA(2,I3)>AB(2,I3)THEN325
322 TF=1:PRINT#2,"Attacker Wpn";I3;"Is Below Breakpoint"
323 PRINT#2,"  Bp =";:PRINT#2,USING"####.##";AB(2,I3);
324 PRINT#2,"  Current Level =";:PRINT#2,USING"####.##";QA(2,I3)
325 NEXTI3
335 FORI3=1TOND:IFQD(I3)>DB(2,I3)THEN340
337 TF=1:PRINT#2,"Defender Wpn";I3;"Is Below Breakpoint"
338 PRINT#2,"  Bp =";:PRINT#2,USING"####.##";DB(2,I3);
339 PRINT#2,"  Current Level =";:PRINT#2,USING"####.##";QD(I3)
340 NEXTI3:IFTF=0THENRETURN
350 PRINT#2,"":PRINT#2,"":PRINT#2,"SUMMARY AT END OF BATTLE"
351 PRINT#2,"":PRINT#2,"Time Elapsed During Battle =";
352 PRINT#2,USING"####.##";TM:PRINT#2,"":GOSUB361
355 CLS:PRINT"Output is in file LANOUT.DO":END
361 PRINT#2," Att Wpn   Breakpoint   Current Level"
363 FORI3=1TONA:PRINT#2,USING"######";I3;
364 PRINT#2,USING"##########.##";AB(2,I3);QA(2,I3):NEXTI3:PRINT#2,""
366 PRINT#2," Def Wpn   Breakpoint   Current Level"
367 FORI3=1TOND:PRINT#2,USING"######";I3;
368 PRINT#2,USING"##########.##";DB(2,I3);QD(I3):NEXTI3:PRINT#2,"":RETURN
```

Figure 5.7 Subroutine To Test For Breakpoints.

criteria to the default criteria. For example, if the operator wants the battle to terminate when A_1 or A_2 reach half their starting strength or when A_1 reaches 60% and A_2 reaches 70% of their starting strength the operator should:

- Put .5 as the individual breakpoints for A_1 and A_2 in the input file and
- Put the program lines shown in Figure 5.8 into the program after line 325.

```
330 IFQA(2,1)/SA(1)>.6ORQA(2,2)/SA(2)>.7THEN335
331 TF=1:PRINT#2,"Special Termination Criterion Met"
```

Figure 5.8 Example Of An Additional Termination Criterion.

If a breakpoint has been reached, then lines 350-368 print the status of both sides at the end of the battle. That end of battle status report includes a header, time elapsed during the battle, and a list of attacking and defending weapon types with their breakpoints and number of survivors. Lines 361-368 are called as a subroutine from line 352 because lines 361-368 are also used to print the summary at the end of each phase and can therefore not terminate the program.

6. Graphics Display Initialization Subroutine, Figure 5.9

```
600 'Set up output screen
610 PRINT"Wpn #    Attacker          Defender"
620 FORI1=1TOMD:PRINTUSING"##";I1
623 TP=2+I1*8
625 IFI1>NATHEN630
627 LINE(18,TP)-(119,TP+4),1,B:BP=18+INT(100*AB(1,I1))
628 LINE(BP-1,TP+1)-(BP,TP+3),1,B
630 IFI1>NDTHEN635
632 LINE(138,TP)-(239,TP+4),1,B:BP=138+INT(100*DB(1,I1))
633 LINE(BP-1,TP+1)-(BP,TP+3),1,B
635 NEXTI1:RETURN
```

Figure 5.9 Graphics Display Initialization Subroutine.

The graphics display consists of a rectangle on the M100 screen for each attacking and defending weapon type. Each rectangle is 100 pixels wide and five pixels high. Each pixel in the horizontal direction represents one percent of the starting strength of the weapon type represented by a particular rectangle. The rectangles are arranged in two columns, one for attacking and one for defending weapon types. This subroutine draws the rectangles, labels the columns "Attacker" or "Defender", puts a vertical line in each box at the breakpoint for that weapon type, and labels the rows of boxes with the weapon type number.

Line 610 prints the header on line one of the screen. Line 620 starts a loop that writes the weapon type number, I1, and prints the rectangles; Line 623 calculates the vertical pixel position, TP, for the top of the rectangles of weapon type I1. Line 625 tests whether to draw a rectangle next to weapon type number I1 in the "Attacker" column. If so, the first statement on line 627 draws the rectangle. The second statement on line 627 calculates the horizontal pixel location in the rectangle of the breakpoint for that weapon type. Line 628 draws a double line at the breakpoint in the rectangle. Lines 630-635 test whether a rectangle should be drawn in the defender column. If so, they draw the rectangle and insert the breakpoint in the same manner as was done in lines 620-628. Line 635 returns control to the main program when there are no more rectangles to be drawn.

7. Updating The Graphical Display, Figure 5.10

```
650 'Update screen output of attackers
653 TP=3+I3*8
655 LINE(OA(I3),TP)-(OA(I3),TP+2),0
656 OA(I3)=18+INT(100*QA(2,I3)/SA(I3))
657 IFOA(I3)>118THENOA(I3)=118:PRINT@(I3*40+2),"*":GOTO659
658 PRINT@I3*40+2," "
659 LINE(OA(I3),TP)-(OA(I3),TP+3),1:RETURN
660 'update screen output of defenders
663 TP=3+I3*8
665 LINE(OD(I3),TP)-(OD(I3),TP+2),0
666 OD(I3)=138+INT(100*QD(I3)/SD(I3))
667 IFOD(I3)>238THENOA(I3)=238:PRINT@I3*40+22,"*":GOTO669
668 PRINT@I3*40+22," "
669 LINE(OD(I3),TP)-(OD(I3),TP+3),1:RETURN.
```

Figure 5.10 Subroutines To Update The Graphical Display.

This section includes two subroutines which update the vertical line in each rectangle which indicates the fraction of survivors for that weapon type. The subroutine in lines 650-659 updates the attacker rectangles; lines 660-669 perform the same function for defender rectangles. Line 653 sets TP, the location of the top pixel of the vertical line for A_{I3}. Line 655 erases the old vertical line, the horizontal pixel position for which was stored in OA(I3). Line 656 calculates the horizontal pixel position for the new vertical line based upon the fraction of the starting strength of A_{I3}'s which currently survives. If the reinforcement rate exceeds the attrition rate and drives the number of survivors over the starting strength for $A_{I3,}$ line 657 holds horizontal position of the vertical line at the 100% level and prints an asterisk next to the corresponding rectangle. If the number of survivors is less than the starting strength, line 658 prints a blank space next to the corresponding rectangle. Line 659 writes the new vertical line to the screen showing the fraction of A_{I3}'s which survive. Lines 660-669 perform the same function for D_j that lines 650-659 perform for A_i.

G. EXAMPLE SIMULATIONS

1. Example #1

The first example uses the input file shown in Figure 5.2 The output file for that simulation is shown in Figure 5.11.

```
STARTING PHASE 1
Status After Phase 1

 Att Wpn   Breakpoint   Current Level
       1      100.00        173.93
       2       50.00         68.55
 Def Wpn   Breakpoint   Current Level
       1       50.00         79.12
       2      100.00        184.53
       3       50.00         89.95

STARTING PHASE 2
Attacker Wpn 2 Is Below Breakpoint
   Bp =  50.00  Current Level =  48.85

SUMMARY AT END OF BATTLE
Time Elapsed During Battle =   7.20
 Att Wpn   Breakpoint   Current Level
       1      100.00        157.29
       2       50.00         48.85
 Def Wpn   Breakpoint   Current Level
       1       50.00         66.24
       2      100.00        174.64
       3       50.00         84.25
```

Figure 5.11 Output File, LANOUT.DO, For Example #1.

2. Comparing The Lanchester and Helmbold Linear Law Equations

Examples 2 and 3 compare the differences between using a traditional Lanchester linear law equation (Example 2) and a Helmbold equation (Example 3). The scenereos for Examples 2 and 3 share the following elements.

- The battle has only one phase with a maximum length of 10 which is broken into 100 intervals.
- Both sides have two weapon types. Each weapon type has a starting strength of 100.
- The breakpoints for all weapon types are 50%.
- All weapon types are linear law weapons. Attrition is calculated using Equation 5.1 for Example 2 and using Equation 5.4 ($\omega = .5$) for Example 3.
- There are no replacements for any weapon type.

The only differences in the scenereos for Examples 2 and 3 are the attrition coefficients.

a. Example #2

The input and output files, LANIN.DO and LANOUT.DO, for Example 2 are in Figure 5.12. Since these attrition rates are for the Lanchester linear law, the dimensionality of the rates for the attacker are (number of attacker casualties) per

(number of attackers) per (number of defenders) per (unit time). The dimensionality of the rates for the defender are the same with the rolls reversed.

Input File:

```
1 2 2
100 100
100 100
.5 .5
.5 .5
1 1
1 1
10 100
0 0
0 0
.00075 .00075
.0005 .0005
.00025 .00025
.00025 .00025
```

Output File:

```
STARTING PHASE 1
Attacker Wpn 1 Is Below Breakpoint
  Bp =  50.00  Current Level =  49.40

SUMMARY AT END OF BATTLE
Time Elapsed During Battle =   5.20

 Att Wpn   Breakpoint   Current Level
     1        50.00        49.40
     2        50.00        62.53

 Def Wpn   Breakpoint   Current Level
     1        50.00        82.17
     2        50.00        82.17
```

Figure 5.12 Input And Output Files For Example #2.

b. Example #3

The input and output files, LANIN.DO and LANOUT.DO, for Example 3 are in Figure 5.13. The attrition rates in this example are for the Helmbold Equation with $\omega = .5$, the Helmbold equivalent of the Lanchester linear dimensionality of the rates for the attacker are (number of attacker casualties) per (attacker)$^{.5}$ per (defender)$^{.5}$ per (unit time). The dimensionality of the rates for the defender are the same with the rolls reversed.

To generate Helmbold coefficients that are comparable to the Lanchester linear law coefficients, the Lanchester coefficients must be adjusted by the difference in the dimensionality, i.e. multiplied by [(number of attackers)(number of defenders)]$^{.5}$. In this example it means multiplying the Lanchester coefficients by 100.

The results of Examples 2 and 3 show good agreement.

- The simulation using the Helmbold equations ended about 21% faster than did the battle using Lanchester equations. The same weapon type reached its breakpoint first in both cases.
- The differences between the number of survivors for other weapon types was quite small.

```
Input File:                    Output File:

1 2 2                          STARTING PHASE 1
100 100                        Attacker Wpn 1 Is Below Breakpoint
100 100                          Bp =  50.00  Current Level =  49.85
.5 .5
.5 .5                          SUMMARY AT END OF BATTLE
.5 .5                          Time Elapsed During Battle =   4.10
.5 .5
10 100                          Att Wpn   Breakpoint   Current Level
0 0                                 1        50.00         49.85
0 0                                 2        50.00         64.67
.0000075 .0000075
.000005 .000005                 Def Wpn   Breakpoint   Current Level
.0000025 .0000025                   1        50.00         82.79
.0000025 .0000025                   2        50.00         82.79
```

Figure 5.13 Input And Output Files For Example #3.

VI. GEOMETRIC PROGRAMMING

A. GENERAL

The program described in this chapter solves nonlinear programming problems in which:

- The objective function is to be minimized.
- The objective function and constraints are posynomials.
- The number of terms,[7] T, minus the number of variables, N, must equal one.
- The coefficients,[8] $c_{m,t}$, of all terms must be strictly positive.
- All components of the vector of decision variables, χ, must be strictly positive at optimality.
- Constraints must have the form of a posynomial on the left hand side that is less than or equal to one.

Geometric programming has the distinctive feature of calculating the optimal value of the objective function before the optimal values of the decision variables are calculated. Geometric programming also produces weights, δ_t, t = 1,2,3,...,T, associated with each term. For example, in applications where the c_t are prices and the objective

---

[7]The number of terms in the objective function and in the constraints.

[8]Two subscripting systems are used throughout this chapter. The first uses the letters *m* and *t* where $t = 1,2,3,...,T_m$ is the number of the term in the mth posynomial. m = 0 refers to the objective function; m = 1,2,... refers to the constraint numbers. T_m is the number of terms in the mth constraint. When problems of only one posynomial are being discussed, the m is omitted. The first system also includes the letter n to identify components of the decision variable vector, χ, where n = 1,2,...,N. The second system numbers the terms without starting again at 1 at the beginning of each constraint. Each term is numbered t', t' = 1,2,...,T' where T' is the number of terms in the objective function and the constraints. The first term of the objective function is denoted by t' = 1. The other terms in the objective function are then numbered from left to right. Then the terms in each constraint in turn are numbered from left to right. For example, in Figure 6.1, t = 1, m = 0 (or t' = 1) refers to $40x_1x_2$ and t = 2, m = 1 (or t' = 5) refers to $.6x_2^{-1}x_3^{-2.3}$.

function minimizes total cost, the δ_t for objective function terms are the proportion of cost that term t contributes to optimal total cost, $f(\chi^*)$. These weights are invariant with respect to the prices, c_t, associated with each term.

1. **Definition Of A Posynomial Function**

The function $f(\chi)$ is posynomial if it has the form

$$f(\chi) = \sum_{t=1}^{T} c_t \, p_t(\chi) \quad \text{where } p_t(\chi) = \prod_{n=1}^{N} x_n^{a_{n,t}} \qquad \text{(eqn 6.1)}$$

where

- T is the number of terms.
- c_t are positive scalar constants.
- χ is the vector of decision variables, $(x_1, x_2, \ldots, x_N)$.
- The only restriction on the exponents, $a_{m,n,t}$, is that they be real numbers.

A posynomial differs from a polynomial in that the coefficients of a posynomial must be strictly positive and its exponents, $a_{n,t}$, need not be positive integers.

An example of a problem meeting these conditions is in Figure 6.1.

$$\text{Min } 40x_1x_2 + 20x_2x_3$$

Subject to:

$$.2x_1^{-1}x_2^{.5} + .6x_2^{-1}x_3^{-2/3} \leq 1$$

$$x_i > 0, \; i=1,2,3$$

Figure 6.1 Geometric Programming Problem In Standard Form.

Geometric programming solves a problem of this kind by solving its dual. When, as specified above, the number of terms minus the number of variables equals one, then the problem has a unique solution. T - N - 1 is by convention called the *degree of difficulty*. If the degree of difficulty is greater than zero, then another nonlinear program must be solved to find the optimal δ_t^*. While this new nonlinear program is frequently easier to solve than the original problem, its solution is beyond the scope of this chapter which is limited to problems with a degree of difficulty of zero.

B. MATHEMATICAL BASIS FOR GEOMETRIC PROGRAMMING

The mathematical basis for geometric programming is summarized in [Ref. 10:pp. 494-522]. and explained in detail in Reference 11. The following explanation is provided for tutorial purposes and is an adaptation of the explanation in [Ref. 10:pp. 496-502]. Notation in this chapter is consistent with that used in Reference 10.

a. Unconstrained Minimization Of Posynomial Functions

Historically, geometric programming has been based upon and took its name from the arithmetic-geometric mean inequality:

$$\sum_{t=1}^{T} v_t\delta_t \geq \prod_{t=1}^{T} v_t^{\delta_t} \quad \text{if } v_t, \delta_t > 0 \text{ and } \sum_{t=1}^{T} \delta_t = 1. \qquad \text{(eqn 6.2)}$$

The equality holds only when $v_1 = v_2 = ,...,v_T$. If u_t is defined as $u_t = v_t\delta_t$, then Equation 6.2 becomes

$$\sum_{t=1}^{T} u_t \geq \prod_{t=1}^{T} (u_t/\delta_t)^{\delta_t} \qquad \text{(eqn 6.3)}$$

The equality holds if $\delta_t = u_t/\sum u_t$. Let u_t be a posynomial term as described in Equation 6.4.

$$u_t = c_t[p_t(\chi)] = c_t\prod_{n=1}^{N} x_n^{a_{n,t}} \qquad \text{(eqn 6.4)}$$

A posynomial function, $f(\chi)$, is given by Equation 6.5.

$$f(\chi) = \sum_{t=1}^{T} u_t \qquad \text{(eqn 6.5)}$$

When the posynomial terms are substituted into Equation 6.3, the inequality becomes

$$\sum_{t=1}^{T} u_t \geq \prod_{t=1}^{T} \{[c_t \prod_{n=1}^{N} x_n^{a_{n,t}}]/\delta_t\}^{\delta_t} \qquad \text{(eqn 6.6)}$$

or

$$\sum_{t=1}^{T} u_t \geq \{\prod_{t=1}^{T} [c_t/\delta_t]^{\delta_t}\}\{\prod_{n=1}^{N} x_n^{\varphi}\} \qquad \text{(eqn 6.7)}$$

where φ is the sum over t of $a_{n,t}\delta_t$.

Since the only restriction on δ_t has been that they be positive and sum to one, they may be chosen such that $\varphi = 0$ for n = 1,2,...,N. If this selection is made, Equation 6.5 becomes

$$f(\chi) = \sum_{t=1}^{T} c_t p_t(\chi) \geq \prod_{t=1}^{T} (c_t/\delta_t)^{\delta_t} \qquad \text{(eqn 6.8)}$$

Since equality holds when $\delta_t = u_t/\sum u_t$, then

$$\min \sum_{t=1}^{T} u_t = \max \prod_{t=1}^{T} (c_t/\delta_t)^{\delta_t} \qquad \text{(eqn 6.9)}$$

$$\text{if} \sum_{t=1}^{T} \delta_t = 1 \text{ and } \sum_{t=1}^{T} a_{n,t}\delta_t = 0 \text{ for n} = 1,2,...,N$$

Therefore, the minimization of the posynomial function is the same as the maximization of the nonlinear function in Equation 6.9 *subject to linear constraints.* The linearity of the constraints means that δ_t^* and χ^* can be computed with linear algebra as explained below, instead of with nonlinear programming. These minimization and maximization problems are duals. Since equality holds if and only if $\delta_t = u_t/\sum u_t$, it follows that the relationship between optimal values of δ and χ is

$$\delta_t^* = \{c_t p_t(\chi^*)\}/f(\chi^*) \text{ or} \qquad \text{(eqn 6.10)}$$

$$\delta_t^* = \{c_t \prod_{n=1}^{N} (x_n^*)^{a_{n,t}}\}/f(\chi^*). \qquad \text{(eqn 6.11)}$$

If the degree of difficulty is zero, then the matrix of exponents, $a_{n,t}$, with another row of 1's appended to the top makes a square matrix. Rows of the exponent matrix correspond to variables and columns correspond to terms. The row of 1's corresponds to the constraint that the sum over t of δ_t equals 1. The δ^* can be obtained by solving a set of T simultaneous linear equations $A\delta^* = b$. The first element of the b vector is 1 and the remaining elements are 0. The optimal value of the objective function, $f(\chi^*)$ can then be obtained by inserting δ_t^* into Equation 6.12. Equation 6.12 is based upon Equation 6.9.

$$f(\chi^*) = \prod_{t=1}^{T} (c_t/\delta_t^*)^{\delta_t^*} \qquad \text{(eqn 6.12)}$$

Finally, the optimal values of the decision variables, x_n, can be determined by solving the set of T equations of the form specified in Equation 6.13.

$$\sum_{n=1}^{N} a_{n,t} \ln(x_n^*) = \ln[f(\chi^*)\delta_t^*/c_t] \quad \text{for } t = 1,2,...,T. \qquad \text{(eqn 6.13)}$$

This set of equations is overconstrained since there are only T-1 decision variables. Therefore, only T-1 equations are required. Solving these T-1 equations simultaneously produces $\rho_n = \ln(x_n^*)$ which are then converted to the optimal values of the decision variables by $x_n^* = e^{\rho_n}$.

b. Inequality Constraints

This section discusses the addition of posynomial inequality constraints to the unconstrained problem discussed above. For notational purposes the objective function and constraints will be numbered m=0,1,2,3,...,M. The objective function is designated m=0, and the constraints are designated m=0,1,2,3,...,M. A primal constrained posynomial would have the form

$$\text{Min} \sum_{t=1}^{T} \{c_{0,t} \prod_{n=1}^{N} x_n^{a_{0,n,t}}\} \qquad \text{(eqn 6.14)}$$

Subject to:

$$f_m(x) = \sum_{t=1}^{T} c_{m,t} \prod_{n=1}^{N} x_n^{a_{m,n,t}} \leq 1 \quad \text{for } m = 1,2,...,M \qquad \text{(eqn 6.15)}$$

where $x_n > 0$, $n = 1,2,3,...,N$. If $\delta_{0,t}$ are the weights for the terms in the objective function, then

$$\delta_{0,t}^* = [c_{0,t}\, p_{0,t}(\chi^*)]/f_0(\chi^*) \quad \text{for } t=1,2,3,...,T_0. \qquad \text{(eqn 6.16)}$$

If λ_m are the Lagrange multipliers associated with constraint m, then

$$\lambda_m = \sum_{t=1}^{T_m} \delta_{m,t} \quad \text{and} \qquad \text{(eqn 6.17)}$$

$$\delta_{m,t}/\lambda_m = c_{m,t}\, p_{m,t}(\chi). \qquad \text{(eqn 6.18)}$$

The dual geometric program is

$$\text{Max} \prod_{m=1}^{M} \prod_{t=1}^{T_m} [c_{m,t}\lambda_m/\delta_{m,t}]^{\delta_{m,t}} \qquad \text{(eqn 6.19)}$$

Subject To:

$$\sum_{t=1}^{T} \delta_{0,t} = 1 \qquad \text{(eqn 6.20)}$$

$$\sum_{m=1}^{M} \sum_{t=1}^{T_m} a_{m,n,t}\, \delta_{m,t} = 0 \text{ for } n = 1,2,3,...,N. \qquad \text{(eqn 6.21)}$$

$$\lambda_m = \sum_{t=1}^{T_m} \delta_{m,t} \qquad \text{(eqn 6.22)}$$

and $\delta_{m,t}, \lambda_m \geq 0$.

The $\delta_{m,t}$ are calculated using Equations 6.20 and 6.21 as a set of simultaneous linear equations. The optimal value of the objective function is calculated by multiplying the unconstrained optimum by $\Pi(\lambda_m)^{\lambda_m}$ as in Equation 6.23.

$$f_0(\chi^*) = \prod_{t=1}^{T} (c_t/\delta_t)^{\delta_t} \prod_{m=1}^{M} (\lambda_m)^{\lambda_m} \qquad \text{(eqn 6.23)}$$

χ^* is calculated using Equations 6.24 and 6.25.

$$\sum_{n=1}^{N} a_{0,n,t} \ln(x_n^*) = \ln(f_0(\chi^*)\delta_{0,t}^*/c_{0,t}) \text{ for } t = 1,2,...,T_0, \text{ and} \qquad \text{(eqn 6.24)}$$

$$\sum_{n=1}^{N} a_{m,n,t} \ln(x_n^*) = \ln(\delta_{m,t}^*/(c_{m,t}\lambda_m^*) \qquad \text{(eqn 6.25)}$$

for $t = 1,2,...,T_0$; $m = 1,2,3,...,M$; and $\delta_{m,t}^* > 0$.

As with the unconstrained problem these equations are linear in $\ln(x_n^*) = \rho_n$ After solving for ρ_n using Equations 6.24 and 6.25 as a set of simultaneous linear equations, the decision variables are calculated using $x_n = e^{\rho_n}$.

C. EXPLANATION OF VARIABLES

- B1(NT+1,NT*2) holds the A matrix in the subroutine which solves simultaneous linear equations of the form Ax = b.
- B2(NT) holds the b vector in the subroutine which solves simultaneous linear equations of the form Ax = b.
- B3(NT,NT-1) stores the exponents of the variables in each term. Each row of the matrix corresponds to a term; each column corresponds to a variable. If a variable is not stated explicitly in a term, then its entry in this matrix is zero.
- CT(3,NC,MN) holds three values for each term. CT(1,m,t) holds the coefficient, $c_{m,t}$. CT(2,m,t) holds the weight, $\delta_{m,t}$, for each term. CT(3,m,t) holds $p_{m,t}(\chi^*)$ for each term. m = 0 refers to terms in the objective function. m = 1,2,...,NC refers to terms in the mth constraint. t = 1,2,...,NT(m) specifies a particular term in the objective function or a constraint.
- FS is the optimal value of the objective function.
- I1,I2, and I3 are loop counters.
- K2,K3,K4,...,K9 are variables in the simultaneous linear equation solving subroutine. This subroutine is documented in Appendix E.
- LM(NC) holds values of λ_m, m = 1,2,...,NC where λ_m is the sum of $\delta_{m,t}$ for the mth constraint. $\lambda_0 = 1$.
- MN is the maximum number of terms in any constraint or the objective function.
- NC is the number of constraints.
- NT(NC) is the number of terms in each constraint.
- NT is the number of terms in the objective function and the constraints.
- NV is the number of decision variables, i.e. the number of components in the vector χ.

D. INPUT

Problem parameters are entered into an input file, GEOIN.DO, before the program is executed. GEOIN.DO must contain the following parameters in the order specified.

- The number of terms, NT, the number of variables, NV, and the number of constraints, NC.
- The number of terms in the objective function, NT(0), and the number of terms in each constraint, NT(m) m = 1,2,...,NC.
- The coefficients of each term, $c_{m,t}$, CT(1,m,t) m = 0,1,2,...,NC, t = 1,2,...,NT(m).

- The matrix of exponents, $a_{n,t}$, for each variable in each term. Row n of the matrix corresponds to the variable x_n, n = 1,2,...,N. Column t of the matrix corresponds to the term $p_{t'}(\chi)$, t' = 1,2,...,T'.

The input file for the problem specified in Figure 6.1 is in Figure 6.2.

```
Input File:
4, 3, 1
2, 2
40, 20
.2, .6
1, 0, -1, 0
1, 1, -.5, -1
0, 1, 0, -.66666666667
```

Problem:

Min $40x_1x_2 + 20x_2x_3$

Subject To:

$.2x_1^{-1}x_2^{.5} + .6x_2^{-1}x_3^{-2/3} \leq 1$

$x_1 > 0$

Figure 6.2 Sample Input File, GEOIN.DO.

E. OUTPUT

The program prints the following output to the screen.

- The optimal dual variables, $\delta_{m,t}^*$.
- The optimal value of the objective function, $f(\chi^*)$.
- The value of each $p_{m,t}(\chi^*)$
- The optimal value of each component of χ, x_n^*.

F. EXPLANATION OF PROGRAM COMPONENTS

A complete program listing is located at Appendix D.

1. Initialization And Input, Figure 6.3

Line 110 opens the input file, GEOIN.DO. Line 120 enters the number of terms, NT, and the number of variables, NV, from GEOIN.DO, and sets K9 equal to NT for use in the simultaneous linear equation solving subroutine. Line 122 checks whether the degree of difficulty is equal to zero. If it is not, an error message is printed and the program ends. Line 130 enters the number of constraints, NC, and dimensions the vector NT(NC), which holds the number of terms in the objective function and in each constraint. Line 140 enters NT(m), m = 0,1,2,...,NC, and computes MN, the maximum over m of NT(m). Line 143 dimensions the matrices required for the program. Line 145 enters the coefficients $c_{m,t}$, placing them in CT(1,m,t).

```
100 'Geometric Programming Program
110 OPEN"GEOIN"FORINPUTAS1
120 INPUT#1,NT,NV:K9=NT
122 IFNT-NV<>1THENPRINT"***ERROR: Degree of Difficulty <> 0":END
130 INPUT#1,NC:DIMNT(NC)
140 MN=0:FORI1=0TONC:INPUT#1,NT(I1):IFNT(I1)>MNTHENMN=NT(I1):NEXTI1
143 DIMCT(3,NC,MN),LM(NC),B1(NT+1,NT*2),B2(NT),B3(NT,NV)
145 FORI1=0TONC:FORI2=1TONT(I1):INPUT#1,CT(1,I1,I2):NEXTI2:NEXTI1
150 FORI1=1TONT(0):B1(1,I1)=1:NEXTI1
155 FORI1=NT(0)+1TONT:B1(1,I1)=0:NEXTI1
160 FORI1=2TONT:FORI2=1TONT:INPUT#1,B1(I1,I2):B3(I2,I1-1)=B1(I1,I2)
162 NEXTI2:NEXTI1
```

Figure 6.3 Initialization and Input.

Lines 150-162 enter elements of the A matrix required to solve the simultaneous linear equations $A\delta = b$ into B1(,). Equations 6.20 and 6.21 are the basis for this set of simultaneous linear equations. Lines 150-155 fill the first row of B1 with ones in columns corresponding to objective function terms and zeros in columns corresponding to other terms. B1(1,t) corresponds to Equation 6.20. Lines 160-162 enter the exponents of variables in each term into B1 and B3. In B1 rows 2 through $NV+1=NT$ correspond to variables x_n and columns 1 through NT correspond to terms $t'=1,2,...,T'$. Storing the exponents in B3 is necessary because the simultaneous linear equation subroutine changes the matrix in B1, and the exponents are required for later calculations. B3 is the transpose of B1 because of the nature of the calculation for which B3 is later recalled.

2. **Calculating The Weights For Each Term, Figure 6.4**

```
170 PRINT"":PRINT"**COMPUTING DELTA'S**"
172 B2(1)=1:FORI1=2TONT:B2(I1)=0:NEXTI1
180 GOSUB9800
200 CLS:I1=1:FORI2=0TONC:FORI3=1TONT(I2):CT(2,I2,I3)=B1(I1,1)
203 PRINT"DELTA(";I2;",";I3;") = ";
204 PRINTUSING"#####.####";CT(2,I2,I3):I1=I1+1:IFI1>5THENGOSUB600
205 NEXTI3:NEXTI2:GOSUB600:CLS
```

Figure 6.4 Calculating $\delta_{m,t'}$.

Line 172 places the simultaneous linear equation b vector into B2. B2(1) = 1 is the right hand side of the constraint in Equation 6.20. The right hand sides of the constraints based upon Equation 6.21 are zero. Line 180 calls the simultaneous linear equation solver in subroutine 9800 which leaves δ^* in B1(t',1), t' = 1,2,...,T'. Lines 200-205 place δ^* into CT(2,m,t) and prints $\delta_{m,t}^*$ to the screen.

3. **The Optimal Objective Function Value, Figure 6.5**

```
210 PRINT"":PRINT"**COMPUTING OPT OBJ FN VALUE**"
212 FORI1=0TONC:LM(I1)=0:FORI2=1TONT(I1):LM(I1)=LM(I1)+CT(2,I1,I2)
214 NEXTI2:NEXTI1
220 FS=1:FORI1=0TONC
222 FORI2=1TONT(I1):FS=FS*(CT(1,I1,I2)/CT(2,I1,I2))^CT(2,I1,I2)
224 NEXTI2:FS=FS*(LM(I1)^LM(I1)):NEXTI1
229 PRINT"":PRINT"F* =";:PRINTUSING"####.###";FS:GOSUB600:CLS
```

Figure 6.5 The Optimal Objective Function Value.

Lines 212-214 compute λ_m which equal the sum over t of $\delta_{m,t}$ for every constraint and the objective function. Lines 220-224 compute the optimal objective function value based upon Equation 6.23. Line 229 prints the optimal objective function value.

4. **Optimal Decision Variable Values, Figure 6.6**

```
230 'Compute optimal x(n)
232 K9=K9-1
234 FORI1=1TOK9:FORI2=1TOK9:B1(I1,I2)=B3(I1,I2):NEXTI2:NEXTI1
236 CC=1:FORI1=0TONC:FORI2=1TONT(I1)
237 CT(3,I1,I2)=(CT(2,I1,I2)/CT(1,I1,I2)/LM(I1))
238 IFI1=0THENCT(3,I1,I2)=CT(3,I1,I2)*FS
239 B2(CC)=LOG(CT(3,I1,I2)):CC=CC+1:NEXTI2:NEXTI1
242 PRINT"P(m,t)* = opt. value of term t, constr. m, divided by its coef."
244 FORI1=0TONC:FORI2=1TONT(I1):PRINT"P(";I1;",";I2;")* =";
246 PRINTUSING"####.####";CT(3,I1,I2):NEXTI2:GOSUB600:NEXTI1:CLS
250 PRINT"":PRINT"** Computing Opt Values Of X(n) **"
260 GOSUB9800
270 CLS:FORI1=1TOK9:PRINT"X*(";I1;") = ";
272 PRINTUSING"######.#####";EXP(B1(I1,1))
273 IFI1>5THENGOSUB600
275 NEXTI1:GOSUB600:END
```

Figure 6.6 Optimal Decision Variable Values.

After δ^* and the optimal value of the objective function, $f_0(\chi^*)$, have been calculated, this section calculates χ^*. The basis for these calculations is Equations 6.24 and 6.25. The section solves a set of simultaneous linear equations $A[\ln(\chi^*)] = b$ and then solves for χ^*.

Since the number of variables is one less than the number of terms, line 232 reduces K9 by one. Line 234 creates the A matrix by putting the matrix of exponents that was stored in B3 into B1.

Lines 236-239 calculate the b vector. CC is a counter in the t′ subscripting system which controls the entry of b vector elements into B2(t′). Line 237 calculates the portion of the right hand side that is common to all terms. Line 238 multiplies that result by $f_0(\chi^*)$ for objective function terms which produces $p_{t'}(\chi^*)$. Line 239 places $\ln[p_{t'}(\chi^*)]$ into B2(t′). Lines 242-246 print $p_{m,t}(\chi^*)$ to the screen. Line 260 calls the simultaneous linear equation subroutine which solves $A[\ln(\chi^*)] = b$. Lines 270-275 print χ^* to the screen and end the program.

5. **Subroutine To Stop Screen Printing, Figure 6.7**

```
600 INPUT"** Hit ENTER To Continue: ";Z9:RETURN
```

Figure 6.7 Subroutine To Stop Screen Printing.

Subroutine 600 is used to interrupt printing loops so that results are not scrolled off the screen before the operator can read them.

6. **Simultaneous Linear Equation Subroutine, Figure 6.8**

This subroutine solves simultaneous linear equations of the form $Ax = b$. K9 is the dimension of the square A matrix and the x and b vectors. This subroutine documented in Appendix E, The Matrix Algebra Program.

G. EXAMPLE PROBLEMS

1. **Example #1**

A design engineer wants to design a cylindrical oil storage tank with a storage capacity of 1000π cubic feet to put on an existing base. If the cost of construction is $\$1/\text{foot}^2$ of tank surface, what are the optimal dimensions of the tank and how much

```
9800 'Simultaneous Linear Equation Subroutine: Ax=b
9815 'Invert Matrix A
9820 FORK7=K9+1TO2*K9:FORK8=1TOK9
9822 IFK7=K8+K9THENB1(K8,K7)=1ELSEB1(K8,K7)=0
9825 NEXTK8:NEXTK7
9830 FORK7=1TOK9
9835 IFB1(K7,K7)*SGN(B1(K7,K7))<1E-8THENGOSUB9910
9840 K2=1/B1(K7,K7):FORK6=1TO2*K9:B1(K7,K6)=B1(K7,K6)*K2:NEXTK6
9842 IFK7=K9THEN9865
9845 FORK8=K7+1TOK9:IFB1(K8,K7)=0THEN9860
9850 K2=-B1(K8,K7)
9855 FORK6=K7TO2*K9:B1(K8,K6)=B1(K8,K6)+(K2*B1(K7,K6)):NEXTK6
9860 NEXTK8:NEXTK7
9865 FORK7=K9TO2STEP-1
9870 FORK8=K7-1TO1STEP-1:IFB1(K8,K7)=0THEN9885
9875 K2=-B1(K8,K7)
9880 FORK6=1TO2*K9:B1(K8,K6)=B1(K8,K6)+(K2*B1(K7,K6)):NEXTK6
9885 NEXTK8:NEXTK7
9890 'Mult A Inverse by b
9894 FORK7=1TOK9:B1(K7,1)=0:FORK8=1TOK9:B1(K7,1)=B1(K7,1)+B1(K7,K8+K9)*B2(K8)
9896 NEXTK8:NEXTK7:RETURN
9900 'Error Routine
9903 IFERL>9700ANDERR=11THENPRINT"!!!ERROR: Matrix Is Not Invertable!!!":END
9905 PRINT"Error Code";ERR;"In Line";ERL:END
9910 'SWITCH ROWS
9915 FORK5=K7+1TOK9:IFB1(K5,K7)*SGN(B1(K5,K7))<1E-8THEN9940
9920 FORK4=1TOK9*2:K3=B1(K7,K4):B1(K7,K4)=B1(K5,K4)
9930 B1(K5,K4)=K3:NEXTK4:RETURN
9940 NEXTK5:PRINT"Error: Matrix Not Invertable":END
```

Figure 6.8 Simultaneous Linear Equation Subroutine.

will it cost? The tank includes the cylindrical siding plus the top. The formulation is in Equations 6.26 and 6.27.

$$\text{Min} \quad \$1(\pi r^2) + \$1(2\pi rh) \qquad \text{(eqn 6.26)}$$

Subject To:

$$\pi r^2 h \geq 1000\pi \quad r,h > 0 \qquad \text{(eqn 6.27)}$$

The objective function and constraint are posynomials, but the constraint is not in the ≤ 1 form required by the program. Putting the constraint in ≤ 1 form results in Equation 6.28. This is a zero degree of difficulty problem with three terms, two variables, one constraint, two terms in the objective function, and one term in the constraint.

$$1000r^{-2}h^{-1} \leq 1 \qquad \text{(eqn 6.28)}$$

The coefficients, $c_{t'}$, $t' = 1,2,3$ are π, 2π, and 1000 respectively. The input file and results of the program are at Figure 6.9.

Input File:	Results:
3, 2, 1	$\delta_1 = 1/3$, $\delta_2 = 2/3$, $\delta_3 = 2/3$
2, 1	
3.142, 6.284, 1000	$f(\chi^*) = \$942.48$
0, 1, -1	
2, 1, -2	Optimal Values for r = 10, h = 10.

Figure 6.9 Input File and Results Of Example #1.

The economic interpretation of δ_1 and δ_2 is that regardless of the price for steel, 1/3 of the cost will be for the top and 2/3 will be for the side. If the top and sides were constructed of different types of steel with different prices, these ratios would not change.

2. **Example #2**

This problem is the example stated in Figure 6.1 The input file is in Figure 6.2. $\delta_{t'}$, $t' = 1,2,3,4$ are .5, .5, .5, and .75 respectively. The optimal value of the objective function is 40. The optimal values of x_n, $n = 1,2,3$ are .5, 1, and 1 respectively.

VII. MATRIX ALGEBRA PROGRAM

A. GENERAL

The matrix algebra program, MATALG, performs the following matrix algebra functions: matrix addition, multiplication, and inversion, scalar multiplication, calculation of determinants, integer exponentiation, and solutions to sets of simultaneous linear equations. MATALG is menu driven. The main menu enables the operator to enter a new matrix, print the answer matrix, or call one of the functions listed above. Menus produced by each function prompt the operator for required input. Matrices may be entered from the M100 keyboard or from a RAM file. Output goes to the M100's screen. Intermediate results may be displayed. Operations are performed in the conventional left to right order in which matrix operations are written out on paper, e.g. $A \times B \times C^{-1}$. However, if the series of operations requires altering that order, the operator may store one matrix for future recall. Matrices may be entered in either the left or right hand position.

B. INPUT.

The matrix input subroutine lets the operator select:

- The position[9] of the matrix. If the new matrix is entered for the left side of the operation, the program automatically places the old left side matrix on the right side.
- Whether the matrix will be entered from the keyboard, the input file MATIN.DO, or from RAM storage.
- Whether the matrix will be scalar multiplied or inverted.

The matrix input routine must be accessed from the main menu to enter the first matrix. From then on, when a two matrix operation is selected from the main menu, the subroutine performing that operation automatically calls the input subroutine for the second matrix.

If the input file, MATIN.DO, is used, it must be created before the program is run. MATIN.DO may contain more than one matrix. Matrices must be preceded in MATIN.DO by their dimensions. An example of an input file is at Figure 7.1.
See the general instructions on input files in Chapter 2.

---

[9]Left or right side of the operation.

```
3 3
1 2 3
9 8 7
4 2 3
2,3
1,2,3
4,5,6
```

Note: The input file to the right contains two matrices:

1 2 3		1 2 3
9 8 7		4 5 6
4 2 3		

Figure 7.1 Sample Input File, MATIN.DO.

When matrices are entered from the keyboard, the operator will be prompted for the matrix dimensions and for each matrix element.

C. OUTPUT

All output goes to the screen of the M100. Output may have up to three digits to the left and four digits to the right of the decimal point. If this configuration is not adequate, the operator may modify the format at the line numbers specified in Figure 7.2 for the corresponding functions.

LINE	*FUNCTION*
1082	Determinant
4075	Solution to Simultaneous Linear Equations
6012	Other Matrix Output

See the instructions for the PRINT USING command in Reference 1.

Figure 7.2 Line Numbers Of Output Formats.

D. DESCRIPTION OF VARIABLES

- A1(3,K,K) holds the current matrices. A1(1,,) = Left side matrix/primary matrix/current intermediate answer matrix. A1(2,,) = Right side/secondary matrix. A1(3,,) = Matrix being stored.
- B1(K,K) holds the answer matrix as it is being calculated.
- C(4) and R(4) are the number of columns/rows in A1 or B1. C(1) and R(1) correspond to A1(1,,). C(2) and R(2) correspond to A1(2,,). C(3) and R(3) correspond to A1(3,,). C(4) and R(4) correspond to B1.
- CC is the row counter in matrix output routine.
- CD and RD are the column and row of element to be changed.

- CH is the selection variable for the main menu.
- DET(2) are the determinants of A1(1,,) and A1(2,,).
- EF is the error Flag. 0 - No terminal error has been made. 1 - A terminal error has been made. A terminal error is one from which the program can not recover.
- FF is the file Flag. 1 - MATIN.DO exists in RAM. 0 - MATIN.DO does not exist in RAM.
- I1, I2, I3, J1, J2, and J3 are loop counters.
- K is the largest dimension of largest matrix to be processed.
- K4-K9 are counters in the simultaneous linear equation subroutine.
- MF is the multiplication/addition flag. 0 - Neither the multiplication nor the addition subroutines are running. 1 - The multiplication subroutine is running. 2 - Addition subroutine is running.
- MI is the matrix indicator. It shows which matrix in A1(1-3) is being operated upon.
- MU is an intermediate multiplier in the determinant and matrix inversion subroutines.
- OF is the output flag. 1 - Send output to screen. 0 - Suppress output to screen.
- SF is the simultaneous linear equation (SLE) flag: 1 - The SLE subroutine is running. 0 - The SLE subroutine is not running.
- XP is the umber of times the matrix will be multiplied times itself in the integer exponentiation subroutine.
- Z9 is a general purpose variable.

E. DESCRIPTION OF PROGRAM COMPONENTS

A complete program listing is located at Appendix E.

1. **Initialization, Figure 7.3**

```
100 CLS:PRINT"":PRINT"     *** MATRIX ALGEBRA PROGRAM ***":PRINT""
105 PRINT"IS INPUT MATRIX, 'MATIN.DO' IN RAM?":INPUT"  0=NO, 1=YES";FF
107 IFFF=1THENOPEN"MATIN"FORINPUTAS1
300 PRINT"**Enter The Single Largest Dimension of"
305 INPUT"The Largest Matrix To Be Processed: ";K
310 DIMA1(3,K,K),B1(K+1,K*2),R(4),C(4),DET(2):MI=1:OF=1:SF=0
```

Figure 7.3 Initialization Section.

Line 100 prints the program title. Line 105 permits the operator to specify whether file MATIN.DO will be used as a source of input and sets the file flag, FF, accordingly. Line 107 opens MATIN.DO for input if it is to be used.

Lines 300-305 require the operator to specify the largest dimension, K, of the largest matrix to be processed. Line 310 dimensions matrices A1 and B1 and vectors R, C, and DET and initializes flags MI, OF, and SF.

2. **Main Menu, Figure 7.4**

This section permits the operator to select the next major operation to be conducted and calls the subroutine performing that operation.

```
501 CLS:EF=0:PRINT"****MATRIX ALGEBRA PROGRAM MENU****"
504 PRINT" 1. Enter Starting Left Side Matrix"
505 PRINT" 2. Matrix Inversion"
506 PRINT" 3. Matrix Addition":PRINT" 4. Matrix Multiplication"
508 PRINT" 5. Simultaneous Linear Equations"
509 PRINT" 6. Print Current Answer Matrix":PRINT" 7. Other Options"
510 INPUT"   **Enter Number: ";CH
512 IFCH=1THENMI=1:Z9=0:GOSUB7006
513 IFCH=2THENMI=1:GOSUB2000
514 IFCH=3THENGOSUB3000
515 IFCH=4THENGOSUB5000
516 IFCH=5THENGOSUB4000
517 IFCH=6THENGOSUB6000
518 IFCH<>7THENGOTO501
520 CLS:PRINT"***MORE CHOICES:":PRINT" 1. Determinant"
524 PRINT" 2. Matrix Integer Exponentiation"
526 PRINT" 3. Store Current Matrix"
530 PRINT" 4. Retrieve Stored Matrix":PRINT" 5. Scalar Multiplication"
532 PRINT" 6. Other Options":INPUT"**Enter Number: "; CH
540 IFCH=1THENMI=1:GOSUB1000
548 IFCH=2THENMI=1:GOSUB7600
549 IFCH=3THENGOSUB8000
550 IFCH=4THENMI=1:GOSUB8200
560 IFCH=5THENMI=1:GOSUB5100
570 GOTO501
```

Figure 7.4 Main Menu.

Line 501 initializes EF and prints the main menu header to the screen. Lines 502-510 print the first screen of options and prompt the operator for a selection. Lines 512-518 call the subroutine selected by the operator or branch to the second screen of options. Lines 520-532 print the second screen of options and prompt the operator for a selection. Lines 540-570 call the subroutine selected by the operator or return to the first screen of options.

3. **Pause Control Subroutine, Figure 7.5**

Lines 700-702 stop the program to permit the operator to view material on the screen and permit continuation by pressing the ENTER button.

```
700 'PAUSE CONTROL
702 INPUT"** Hit ENTER To Continue";Z9:RETURN
```

Figure 7.5 Pause Control Subroutine.

4. **Modifying the Secondary Matrix, Figure 7.6**

```
800 'INTERMEDIATE MODIFICATIONS
810 PRINT"**Modify The 2nd Matrix?"
812 INPUT"   0=No, 1=Invert, 2=Scalar Multiply: ";Z9
815 IFZ9=0THENRETURN
820 MI=2:IFZ9=1THENGOSUB2000ELSEGOSUB5100
825 GOTO810
```

Figure 7.6 Modifying The Secondary Input Matrix.

This subroutine permits the operator to invert the second matrix of a two matrix operation or multiply that matrix by a scalar. Lines 810-812 print the options to the screen and prompt the operator for a selection. Line 815 causes a return without the matrix being modified if appropriate. Line 820 sets the matrix indicator, MI, to two and calls the matrix inversion or scalar multiplication subroutine. Line 825 starts the subroutine again, permitting the operator to select another option.

5. **Determinant Calculation, Figure 7.7**

Lines 1005-1008 test whether the matrix is square and print an error message if it is not. Line 1010 copies the matrix to be inverted into B1 where the calculations will be conducted. Line 1020 initializes the value of the determinant as one and the row counter, I1.

Line 1021 checks whether the diagonal element in the current row, R_c, is zero. If so, then the row switching subroutine is called. If all the rows below R_c have 0's in column I1 then the determinant is zero. If a non-zero element can be found below R_c in column I1 then that row is switched with R_c and the determinant is multiplied by -1. Line 1022 tests for a terminal error from the row switching subroutine. Line 1023 multiplies the determinant by diagonal element I1 and branches to the end of the

```
1000 'CALC DETERMINANT
1005 IFR(MI)=C(MI)THEN1010
1007 PRINT"ERROR: Number of rows/columns not equal:"
1008 PRINT"       MATRIX IS NOT INVERTABLE!":GOSUB700:EF=1:RETURN
1010 FORI1=1TOR(MI):FORI2=1TOC(MI):B1(I1,I2)=A1(MI,I1,I2):NEXTI2:NEXTI1
1020 DET(MI)=1:FORI1=1TOR(MI)
1021 IFB1(I1,I1)*SGN(B1(I1,I1))<1E-10THENGOSUB1900ELSE1023
1022 IFEF=1THEN1008
1023 DET(MI)=DET(MI)*B1(I1,I1):IFI1=R(MI)THEN1080
1025 FORI3=1TOC(MI):B1(I1,I3)=B1(I1,I3)/B1(I1,I1):NEXTI3
1030 FORI2=I1+1TOR(MI):IFB1(I2,I1)=0THEN1060
1040 FORI3=I1TOC(MI):B1(I2,I3)=B1(I2,I3)-(B1(I2,I1)*B1(I1,I3)):NEXTI3
1060 NEXTI2:NEXTI1
1080 IFOF<>1THENRETURN
1081 PRINT"**Det. Of Matrix ";MI;" Is: ";
1082 PRINTUSING"######.#####";DET(MI):GOSUB700
1090 RETURN
```

Figure 7.7 Determinant Calculation.

subroutine if the R_c is the last row. Line 1025 divides all elements in R_c by the diagonal element in R_c. Lines 1030-1060 update the elements in R_c and below in in column I1 and to the left. Line 1080 tests whether output is to be printed to the screen. If not, the subroutine ends. If so, lines 1081-1082 print the determinant.

6. **Row Switching Subroutine, Figure 7.8**

```
1900 'SWITCH ROWS
1910 FORJ=I1+1TOR(MI):IFB1(J,I1)*SGN(B1(J,I1))<1E-10THEN1940
1920 FORJ1=1TOC(MI)*2:TE=B1(I1,J1):B1(I1,J1)=B1(J,J1)
1930 B1(J,J1)=TE:NEXTJ1:GOTO1950
1940 NEXTJ:EF=0:RETURN
1950 DET(MI)=-DET(MI):RETURN
```

Figure 7.8 Row Switching Subroutine.

This subroutine is called when the determinant or inversion subroutines try to pivot on a row, R_{I1}, with a zero in the main diagonal element. The subroutine looks for the first row below R_{I1} that has a nonzero element in column I1.[10] Line 1910 searches the rows below R_{I1} for a row with a nonzero element in the appropriate

[10]The same column as the zero on the main diagonal in R_{I1}.

column.[11] Lines 1920-1930 switch the elements of the rows using TE as an intermediate storage variable. If none of the rows below R_{II} have a nonzero element in the appropriate column, then the matrix is not invertable and EF is set to one in line 1940. If a row switch was made, the determinant changes sign in line 1950.

7. **Matrix Inversion Subroutine, Figure 7.9**

```
2000 'MATRIX INVERSION
2010 OF=0:GOSUB1000:IFDET(MI)*SGN(DET(MI))>1E-10OREF=1THEN2017
2015 PRINT"*ERROR: Determinant=0. MATRIX NOT INVERTABLE!":GOSUB700:EF=1
2017 IFEF=1THENRETURN
2020 FORI1=1TOR(MI):FORI2=1TOC(MI):B1(I1,I2)=A1(MI,I1,I2):NEXTI2:NEXTI1
2030 FORI1=C(MI)+1TO2*C(MI):FORI2=1TOR(MI)
2032 IFI1=I2+R(MI)THENB1(I2,I1)=1ELSEB1(I2,I1)=0
2035 NEXTI2:NEXTI1
2040 FORI1=1TOC(MI)
2045 IFB1(I1,I1)*SGN(B1(I1,I1))<1E-10THENGOSUB1900ELSE2055
2046 IFEF=1THEN2015
2055 MU=1/B1(I1,I1):FORI3=1TO2*C(MI):B1(I1,I3)=B1(I1,I3)*MU:NEXTI3
2057 IFI1=C(MI)THEN2080
2060 FORI2=I1+1TOR(MI):IFB1(I2,I1)=0THEN2075
2065 MU=-B1(I2,I1)
2070 FORI3=I1TO2*C(MI):B1(I2,I3)=B1(I2,I3)+(MU*B1(I1,I3)):NEXTI3
2075 NEXTI2:NEXTI1
2080 FORI1=C(MI)TO2STEP-1
2100 FORI2=I1-1TO1STEP-1:IFB1(I2,I1)=0THEN2130
2110 MU=-B1(I2,I1)
2120 FORI3=1TO2*C(MI):B1(I2,I3)=B1(I2,I3)+(MU*B1(I1,I3)):NEXTI3
2130 NEXTI2:NEXTI1
2140 FORI1=1TOC(MI):FORI2=1TOR(MI)
2145 A1(MI,I2,I1)=B1(I2,I1+C(MI)):NEXTI2:NEXTI1
2190 OF=1:RETURN
```

Figure 7.9 Matrix Inversion Subroutine.

The matrix inversion subroutine places the matrix to be inverted, μ, and an identity matrix, I, into B1. Each row of B1 holds a row of μ and the corresponding row from I. Elementary row operations are conducted on μ in B1 to change that portion of B1 to an identity matrix. The same elementary row operations are conducted on the portion of B1 that started as an identity matrix. When the portion of B1 which started as μ is changed to I, then the portion of B1 which started as I becomes μ^{-1}.

[11]The decision rule actually looks for an element outside the range 0 ± 10^{-10}.

Line 2010 stops intermediate results from being printed to the screen by setting OF to zero. Line 2010 also calls the determinant calculation subroutine and tests whether the determinant is equal to zero. If the determinant equals zero, then lines 2015-2017 print an error message, set the error flag, and terminate the inversion subroutine. Line 2020 copies μ into B1. Lines 2030-2035 place an identity matrix with the same dimensions as μ into B1 with μ.

Lines 2040-2075 Line 2045 checks whether the $\mu_{I1,I1}$ is zero. If so, then the row switching subroutine is called. If the row switching subroutine can not find a row for which element I1 does not equal zero, then the matrix is not invertable and line 2046 branches to the error message. Line 2055 calculates the constant, MU,[12] and multiplies row I1 by MU. Since lines 2060-2075 do not apply to the last row of μ, line 2057 branches around them if I1 points to the last row.

Lines 2060-2075 perform the elementary row operations to change to zero the elements of column I1 that are in rows below row i1. Lines 2080-2130 perform the elementary row operations which change to zero the elements of μ above the main diagonal. Lines 2140-2145 copy the inverted matrix from B1 back to the appropriate section of A1. Line 2190 turns the output back on by resetting OF and terminates the subroutine with a return.

8. **Matrix Addition, Figure 7.10**

```
3000 'MATRIX ADDITION
3010 MF=2:GOSUB7000:GOSUB800:IFEF=1THENRETURN
3015 FORI1=1TOR(1):FORI2=1TOC(1):A1(1,I1,I2)=A1(1,I1,I2)+A1(2,I1,I2)
3020 NEXTI2:NEXTI1:GOSUB6000:MF=0:RETURN
```

Figure 7.10 Matrix Addition Subroutine.

Line 3010:

- Sets MI = 2 indicating to the input subroutine that it is being called from the matrix addition subroutine.
- Calls subroutine 7000 to enter the second matrix.

[12]Multiplying row I1 by MU makes main diagonal element $\mu_{I1,I1}$ equal to one, i.e. its identity matrix value.

- Calls subroutine 800 to permit the operator to invert the second matrix or multiply it by a scalar.
- Evaluates whether a terminal error was made in either subroutine 7000 or 800 and, if so, terminates the matrix addition subroutine.

Lines 3015-3020 add the elements of A1(1,,) and A2(2,,). Line 3020 also calls subroutine 6000, printing the answer, resets MF = 0, and terminates the matrix addition subroutine.

9. **Simultaneous Linear Equations, Figure 7.11**

```
4000 'SIMULTANEOUS LINEAR EQUATIONS
4010 CLS:PRINT"**Solves Ax=b.  Choices:":PRINT" 1. Enter b Vector"
4012 PRINT" 2. Change An Element In Matrix A"
4013 PRINT" 3. Solve Current Ax=b"
4014 PRINT" 4. Return":INPUT" * Select A Number: ";CC
4020 IFCC=1GOTO4040
4022 IFCC=2GOTO4050
4024 IFCC=3GOTO4060
4026 IFCC=4THEN RETURN
4035 GOTO 4000
4040 MI=2:R(2)=C(1):C(2)=1:GOSUB7040:GOTO4000
4050 INPUT"**Row, Column Of Matrix A To Be Changed: ";RD,CD
4052 PRINT" - Enter Row";RD;", Column";CD;":";:INPUTA1(1,RD,CD):GOTO4000
4060 MI=1:SF=1:OF=0:GOSUB8000:GOSUB2000
4064 IFEF=0THEN4070
4065 PRINT"*Solution Not Uniquely Determinable":GOSUB700:RETURN
4070 GOSUB5000:CC=0:FORI1=1TOR(2):CC=CC+1:PRINT"x(";I1;") = ";
4075 PRINTUSING"#####.####";B1(I1,1):IFCC>6THENGOSUB700:CC=0
4080 NEXTI1:GOSUB700:SF=0:GOSUB8200:GOTO4000
```

Figure 7.11 Simultaneous Linear Equation Solving Subroutine.

This subroutine solves sets of linear equations of the form Ax = b where A is m by m matrix of rank m and x and b are vectors of length m. Matrix A must be entered as the primary matrix before this subroutine is called. The subroutine prompts the operator to enter b. The subroutine also permits the operator to change individual elements of the A matrix.

Lines 4010-4014 print a header and a menu of options and prompt the operator to select an option. Lines 4020-4035 transfer control to execute the option selected. Line 4040 sets MI = 2, indicating that b will be stored in A1(2,,), dimensions the b vector, and calls subroutine 7040 to input b. Line 4050 prompts the operator for the row and column of the element in A to be changed. Line 4052 prompts the operator to enter the new value for that element.

Lines 4060-4080 solve the system of equations. Line 4060 sets MI, SF, and OF and calls subroutines which store, then invert, the A matrix. Lines 4064-4065 print an error message if the A matrix is not invertable. Lines 4070-480:

- Multiply A^{-1} by b producing the solution vector, x.
- Print the solution vector.
- Reset SF = 0.
- Retrieve the stored A matrix.

10. **Matrix Multiplication Subroutine, Figure 7.12**

```
5000 'MATRIX MULT
5010 MF=1:IF SF=1THEN5020
5015 MI=2:GOSUB7000:GOSUB800:IFEF=1THENRETURN
5020 R(4)=R(1):C(4)=C(2):FORI1=1TOR(4):FORI2=1TOC(4):B1(I1,I2)=0
5022 FORI3=1TOC(1):B1(I1,I2)=A1(1,I1,I3)*A1(2,I3,I2)+B1(I1,I2)
5024 NEXTI3:NEXTI2:NEXTI1:MF=0
5050 IF SF=0THENGOSUB7500:GOSUB6000
5060 RETURN
```

Figure 7.12 Matrix Multiplication Subroutine.

Line 5010 sets MF[13] and branches to avoid the input subroutines in line 5015 if SF equals one.[14] Line 5015 calls the subroutines to enter and modify the second matrix. Lines 5020-5024 set the dimensions B1 and perform the multiplications and additions required to place the answer matrix in B1. If SF equals zero,[15] then line 5050 calls subroutines which copy the answer from B1 to A1(1,,) and print the answer matrix.

11. **Scalar Multiplication Subroutine, Figure 7.13**

This subroutine multiplies matrix A1(MI,,) by a scalar, SM. Lines 5110-5115 prompt the operator to enter the scalar and conduct the multiplication.

---

[13]MF = 1 indicates that matrix multiplication is being performed.

[14]That is, if the matrix multiplication subroutine is called from the simultaneous linear equation subroutine.

[15]That is, this subroutine is not being called from the simultaneous linear equation subroutine.

```
5100 'SCALER MULT
5110 INPUT"Enter Scalar Multiplier:";SM:FORI1=1TOR(MI):FORI2=1TOC(MI)
5115 A1(MI,I1,I2)=A1(MI,I1,I2)*SM:NEXTI2:NEXTI1:RETURN
```

Figure 7.13 Scalar Multiplication Subroutine.

12. **Subroutine To Print A1(1,,), Figure 7.14**

```
6000 'PRINT OUTPUT MATRIX
6010 PRINT" ** Current Answer Matrix:":CC=0:FORI1=1TOR(1):CC=CC+1
6012 FORI2=1TOC(1):PRINTUSING"####.####";A1(1,I1,I2);:NEXTI2:PRINT""
6050 IFCC=3THENGOSUB700:CC=0
6070 NEXTI1:GOSUB700:RETURN
```

Figure 7.14 Subroutine To Print The Primary Matrix.

Lines 6010-6070 print a header and then print A1(1,,). The matrix is printed up to three rows at a time.

13. **Matrix Input Subroutine, Lines 7000-7050**

a. Input Matrix Configuration, Figure 7.15

The section in Figure 7.15 prompts the operator to specify:

- Whether the matrix to be entered will go on the left or right hand side of the operation.
- Whether the matrix will be entered from the keyboard, MATIN.DO, or retrieved from RAM storage. The dimensions of the incoming matrix are entered from the appropriate source.

Line 7001 prompts the operator to specify whether the incoming matrix will be on the left or right side of the operation. If the incoming matrix is to be on the left side, lines 7003-7004 move the matrix in A1(1,,) to A1(2,,). Lines 7006-7008 print an appropriate header. Lines 7009-7011 print the source options for the incoming matrix. The option to enter a matrix from MATIN.DO will be printed only if MATIN.DO has been created (FF = 1) and the end of MATIN.DO has not been reached (EOF(1) = 0). Lines 7012-7013 transfer control to enter the matrix from the appropriate source. Lines 7014-7018 enter the dimensions of the new matrix from the appropriate source.

```
7000 'MATRIX INPUT
7001 CLS:PRINT"":PRINT"Will This Matrix Be On:":INPUT" 0=Left, 1=Right";Z9
7002 IFZ9=1THEN7006
7003 R(2)=R(1):C(2)=C(1):FORI1=1TOR(1):FORI2=1TOC(1):A1(2,I1,I2)=A1(1,I1,I2)
7004 NEXTI2:NEXTI1:MI=1
7006 CLS:IFMI=2THEN7008
7007 PRINT"**Choices For Left Hand Matrix":GOTO7009
7008 PRINT"**Choices For Right Hand Matrix:"
7009 PRINT" 1. Enter Matrix From Keyboard"
7010 PRINT" 2. Retrieve Stored Matrix":IFFF<>1THEN7012
7011 IFEOF(1)=0THENPRINT" 3. Enter Matrix From MATIN.DO"
7012 INPUT"**Enter A Number: ";Z9:IFZ9=1THEN7015
7013 IFZ9=3THEN7018
7014 R(MI)=R(3):C(MI)=C(3):GOTO7020
7015 PRINT" **Enter The Rows, Columns"
7017 INPUT"In The Next Matrix: ";R(MI),C(MI):GOTO7020
7018 INPUT#1,R(MI),C(MI)
```

Figure 7.15 Input Matrix Configuration.

b. Detection Of Dimensioning Errors, Figure 7.16

```
7020 IF MF<>1THEN7030
7021 IFR(2)=C(1)THEN7030
7022 PRINT"**ERROR: Columns in LEFT MATRIX =";C(1)
7024 PRINT"       Rows In Right Matrix =";R(2)
7026 PRINT"These Must Be Equal For Matrix Mult!!":GOSUB700:EF=1:GOTO7006
7030 IF MF<>2THEN7035
7031 IF(R(1)=R(2)ANDC(1)=C(2))THEN 7035
7032 PRINT"**ERROR:Dimensions For Both Input"
7034 PRINT"Matrices Must Be Equal!!":GOSUB700:EF=1:GOTO7006
```

Figure 7.16 Detection Of Dimensioning Errors.

If the matrix being entered is the second matrix in a matrix multiplication operation, then lines 7020-7026 check whether the number of columns in the left matrix is equal to the number of rows in the right matrix. If not, then an error message is printed and control is transfered to the beginning of the matrix input subroutine. If the matrix being entered is the second matrix in a matrix addition operation, then lines 7030-7034 check whether the dimensions of the left and right matrices are the same. If not, then an error message is printed and control is transfered to the beginning of the matrix input subroutine.

c. Matrix Input Section, Figure 7.17

```
7035 IfZ9=2THENGOSUB8200:RETURN
7036 IFZ9=3THEN7050
7037 PRINT" **Fill Matrix Row By Row:":PRINT""
7040 FORI1=1TOR(MI):FORI2=1TOC(MI)
7042 PRINT"-Enter Row";I1;"And Column";I2;":";
7044 INPUTA1(MI,I1,I2):NEXTI2:PRINT"":NEXTI1:RETURN
7050 FORI1=1TOR(MI):FORI2=1TOC(MI):INPUT#1,A1(MI,I1,I2):NEXTI2:NEXTI1:RETURN
```

Figure 7.17 Matrix Input Section.

If the incoming matrix is to be retrieved from RAM storage, line 7035 calls the appropriate subroutine. Line 7037-7044 enter the incoming matrix from the keyboard. If the incoming matrix is to be entered from MATIN.DO then line 7036 transfers control to 7050 which performs the entry.

14. **Copy B1 Into A1(1,,), Figure 7.18**

```
7500 'COPY B1 INTO A1(1,,)
7510 R(1)=R(4):C(1)=C(4):FORI1=1TOR(1):FORI2=1TOC(1)
7512 A1(1,I1,I2)=B1(I1,I2):NEXTI2:NEXTI1:RETURN
```

Figure 7.18 Subroutine to copy B1 into A1(1,,).

Lines 7510-7512 dimension a1(1,,) and copy B1 into A1(1,,).

15. **Matrix Integer Exponentiation, Figure 7.19**

```
7600 'MATRIX INTEGER EXPONENTIATION
7610 CLS:PRINT"":INPUT"**Enter Integer Exponent > 2: ";XP
7620 R(2)=R(1):C(2)=C(1):FORI1=1TOR(1):FORI2=1TOC(2)
7622 A1(2,I1,I2)=A1(1,I1,I2):NEXTI2:NEXTI1
7630 SF=1:FOREX=2TOXP:GOSUB5020:GOSUB7500:NEXTEX:GOSUB6000:SF=0:RETURN
```

Figure 7.19 Matrix Integer Exponentiation Subroutine.

This subroutine raises the primary matrix to an integer power greater than or equal to two. Line 7610 prompts the operator to enter an exponent, XP. Lines 7620-7622 dimension A1(2,,) and copy A1(1,,) into A2(2,,). Line 7630:

- Sets SF = 1. This suppresses printing of intermediate results.
- Performs XP-1 matrix multiplications.
- Prints the final result.

16. **Storage and Retrieval Subroutines, Figure 7.20**

```
8000 'STORE A1(1,,)
8010 R(3)=R(1):C(3)=C(1):FORI1=1TOR(3):FORI2=1TOC(3)
8012 A1(3,I1,I2)=A1(1,I1,I2):NEXTI2:NEXTI1:RETURN
8200 'RETRIEVE THE STORED MATRIX
8210 R(MI)=R(3):C(MI)=C(3):FORI1=1TOR(MI):FORI2=1TOC(MI)
8212 A1(MI,I1,I2)=A1(3,I1,I2):NEXTI2:NEXTI1:RETURN
```

Figure 7.20 Storage and Retrieval Subroutines.

Lines 8010-8012 dimension A1(3,,) and store A1(1,,) in A1(3,,). Lines 8210-8212 dimension A1(1,,) and retrieve A1(1,,) from A1(3,,).

F. SIMULTANEOUS LINEAR EQUATION SUBROUTINE, *FIGURE 7.21*

Many programs require a simultaneous linear equation solver. Often these programs compute the A matrix as part of the program and use the results in subsequent calculations. The following subroutine may be inserted in other programs without requiring the loading of the entire matrix algebra program.

This subroutine follows the same algorithm as the simultaneous linear equation subroutine in the Matrix Algebra Program. K9 is the dimension of the A matrix. B1 holds the A matrix; B2 holds the b vector. The x vector is returned in the first column of B1. If the A matrix is to be used later, it must be stored somewhere other than B1 since the A matrix in B1 is changed to an identity matrix by this subroutine. Instead of testing for a zero determinant, the subroutine uses the error identification subroutine 9900 to determine if the A matrix is not invertable. Variables K2-K9 are used to avoid conflict with other counters.

```
9800 'Simultaneous Linear Equation Subroutine: Ax=b
9802 DIM B1(K9+1,K9*2),B2(K9):'B1 = A matrix; B2 = b vector
9805 'Input from SLEIN.DO; Set MAXFILES in main program
9806 OPEN"SLEIN"FORINPUTAS9
9807 FORK8=1TOK9:FORK7=1TOK9:INPUT#9,B1(K8,K7):NEXTK7:NEXTK8
9808 FORK8=1TOK9:INPUT#9,B2(K8):NEXTK8
9815 'Invert Matrix A
9820 FORK7=K9+1TO2*K9:FORK8=1TOK9
9822 IFK7=K8+K9THENB1(K8,K7)=1ELSEB1(K8,K7)=0
9825 NEXTK8:NEXTK7
9830 FORK7=1TOK9
9835 IFB1(K7,K7)*SGN(B1(K7,K7))<1E-8THENGOSUB9910
9840 FORK6=1TO2*K9:B1(K7,K6)=B1(K7,K6)/B1(K7,K7):NEXTK6
9842 IFK7=K9THEN9865
9845 FORK8=K7+1TOK9:IFB1(K8,K7)=0THEN9860
9855 FORK6=K7TO2*K9:B1(K8,K6)=B1(K8,K6)-(B1(K8,K7)*B1(K7,K6)):NEXTK6
9860 NEXTK8:NEXTK7
9865 FORK7=K9TO2STEP-1
9870 FORK8=K7-1TO1STEP-1:IFB1(K8,K7)=0THEN9885
9880 FORK6=1TO2*K9:B1(K8,K6)=B1(K8,K6)-(B1(K8,K7)*B1(K7,K6)):NEXTK6
9885 NEXTK8:NEXTK7
9890 'Mult A Inverse by b
9892 PRINT"** Sim Lin Eq Solution:  X(i) ="
9894 FORK7=1TOK9:B1(K7,1)=1:FORK8=K9+1TO2*K9:FORK6=1TOK9
9896 B1(K7,1)=B1(K7,1)+B1(K7,K8)*B2(K6):NEXTK6:NEXTK8
9898 PRINTUSING"####.###";B1(K7,1):NEXTK7:PRINT"":RETURN
9900 'Error Routine
9903 IFERL>9700ANDERR=11THENPRINT"!!!ERROR: Matrix Is Not Invertable!!!":END
9905 PRINT"Error Code";ERR;"In Line";ERL:END
9910 'SWITCH ROWS
9915 FORK5=K7+1TOK9:IFB1(K5,K7)*SGN(B1(K5,K7))<1E-8THEN9940
9920 FORK4=1TOK9*2:K3=B1(K7,K4):B1(K7,K4)=B1(K5,K4)
9930 B1(K5,K4)=K3:NEXTK4:RETURN
9940 NEXTK5:PRINT"Error: Matrix Not Invertable":END
```

Figure 7.21 Simultaneous Linear Equation Subroutine.

VIII. NUMERICAL DOUBLE INTEGRATION PROGRAM

A. GENERAL

This program numerically integrates functions of one or two variables using Simpson's Rule with a Romberg extrapolation to improve accuracy. The Romberg extrapolation is described in [Ref. 12:pp.250-276]. Using the Romberg extrapolations allows the operator to specify an acceptable error. The program conducts extrapolations until the error of the numerical estimate is below that specified tolerance.

The operator may interactively change the function being integrated, the limits of integration, or the Romberg tolerance. This program is also written as a subroutine. However, in the subroutine the function being integrated, the limits of integration, and the tolerance may not be interactively changed.

B. INPUT

All input is entered from the keyboard of the M100. When the program begins, a menu appears which allows the operator to select whether the function to be integrated, the limits of integration, or the Romberg tolerance will be changed.

When the operator selects an input to be changed, the program calls the edit function for the applicable lines in the program. The edit function terminates the running of the program. The operator should change only the right hand side of the input equations. After changing the lines required, the operator will hit the F8 button on the M100. This puts the M100 back in the BASIC mode. The operator must then enter RUN or hit the F4 button to run the program again. If the operator wants to change another input, he should select another input from the menu and repeat the process.

1. Changing The Function, f(x,y), To Be Integrated

Enter zero at the main menu. When lines 1285-1288 appear, change the right hand side of the equation on line 1286. If the equation is too long for one line, then:

- Calculate a partial function value on line 1286, assigning it to F.
- Add a line 1287 assigning the final function value to F and including the partial function value from line 1286 in the right hand side of the equation on line 1287.

For example, if f(x,y) = $(x^2+y^5+7) * e^{x^{2*}y^3}$, then the function might be broken down as indicated in Figure 8.1.

```
1286 F=EXP(X^2*Y^3)
1287 F=(X^2+Y^5+7)*F
```

Figure 8.1 Example Of Function To Be Integrated.

Up to two independent variables, X and Y, may be used in the equation. The operator must ensure that f(x,y) is formulated with X as a variable for which constant limits of integration can be specified. After f(x,y) is entered, depress F8, then F4 to return to the main menu.

2. **Changing The Limits Of Integration**

Enter one at the main menu. When lines 1291-1298 appear, change the right hand side of the equations on lines 1293-1298 as desired. The upper and lower limits of integration for X, XUPPER and XLOWER, must be constants. The upper and lower limits of integration for Y, YUPPER and YLOWER, may be constants or given in terms of X. Do not alter the return statement at line 1295. After the limits of integration are entered, depress F8, then F4 to return to the main menu.

3. **Changing the Romberg Tolerance**

The operator should enter two from the main menu and enter the new tolerance when prompted.

4. **Using The Program For Single Integration**

Although the program is written for double integration, single integration may be calculated using the following steps.

- Set the function to be integrated equal to one, i.e. line 1286 will be F = 1.
- In lines 1296-1297 set YUPPER = f(X) and YLOWER = 0.
- In lines 1293-1294 set XUPPER and XLOWER as the constant limits of X between which the function YUPPER = f(X) is to be integrated.

For example, for $\int_0^2 x^2\,dx$, the corresponding limits of integration in lines 1293-1297 would be as indicated in Figure 8.2.

```
1293 XUPPER=2
1294 XLOWER=0
1295 RETURN
1296 YUPPER=X^2
1297 YLOWER=0
```

Figure 8.2 Example of Limits Of Integration.

C. OUTPUT

The estimated value of the integral is printed to the screen with its tolerance error. If the program generated a tolerance error that was less than the tolerance specified in the input, then that tolerance is printed. If the program could not generate an estimate within the specified tolerance, then a message to that effect is printed to the screen.

D. EXPLANATION OF VARIABLES

- A2(6,6) is the matrix holding Romberg extrapolation values.
- DX and DY are the widths of intervals (XU-XL)/N and (YU-YL)/N respectively.
- F is the value of f(x,y) to be integrated at a particular point.
- J1 through J9 are loop counters.
- N is the number of intervals into which the distances XU-XL and YU-YL are divided.
- SS is the Simpson's Rule sum specified in equation 8.1. In equation 8.1 $f_i = f(x,y|x=X)$, $YL \le y_i \le YU$, $i = 1,2,3,...,n+1$. *n* is the number of intervals into which the distance YU-YL has been divided. n must be a positive, even integer.

$$\text{Simpsons's Rule Sum} = f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + ,..., + 4f_n + f_{n+1} \qquad \text{(eqn 8.1)}$$

- TL is the user specified tolerance.
- XLOWER or XL is the lower limit of integration for X.
- XUPPER or XU is the upper limit of integration for X.
- YLOWER or YL is the lower limit of integration for Y.
- YUPPER or YU is the upper limit of integration for Y.
- Z9 is a selection variable.

E. EXPLANATION OF PROGRAM COMPONENTS

A complete program listing is located at Appendix F.

1. Initialization, Figure 8.3

```
1200 'Numberical Double Integration:Steven H. Cary:24 Aug 86
1201 DIMA2(6,6):TL=.001
```

Figure 8.3 Initialization Section.

Line 1201 dimensions the matrix holding the Romberg extrapolations and sets the default tolerance to .001.

2. Option Selection, Figure 8.4

```
1205 CLS:PRINT"":PRINT"       ** Double Integration **"
1206 PRINT"           Romberg Algorithm"
1210 PRINT"0=Edit Function To Be Integrated."
1211 PRINT"1=Edit Limits Of Integration."
1213 PRINT"2=Edit Tolerance; Current Tol.=";TL
1215 PRINT"3=Calculate Integral ":INPUT"Enter 0, 1, 2, or 3:";Z9
1216 IFZ9=0THENEDIT1285-1288
1217 IFZ9=1THENEDIT1291-1298
1218 IFZ9=2THENPRINT"":INPUT"Tolerance=";TL:GOTO1205
```

Figure 8.4 Option Selection Section.

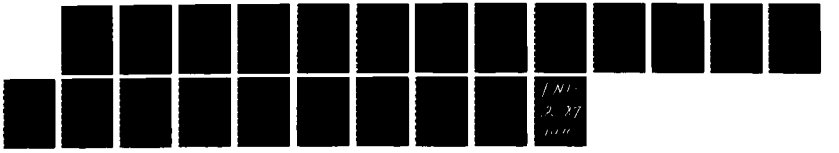
Lines 1205-1218 print a menu which permits the operator to change f(x,y), the limits of integration, or the tolerance, or to calculate the integral. If f(x,y) or the limits of integration are to be changed, then lines 1216 or 1217 activate the editor for the appropriate program lines. The editor terminates program execution, thereby requiring that the program be executed after editing. If the tolerance is to be changed, then line 1218 prompts the operator to update TL and redisplays the menu.

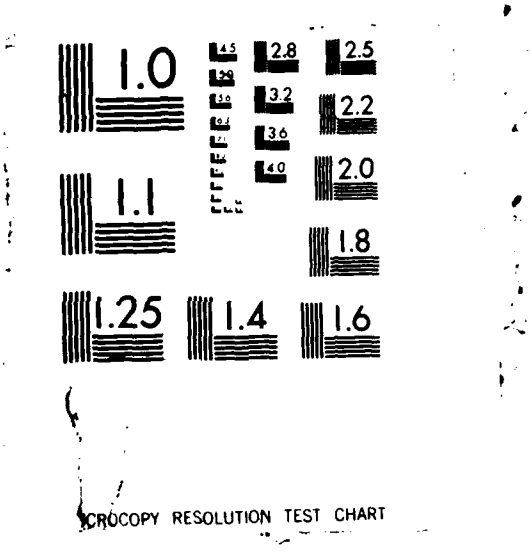
3. Integration Calculation, Figure 8.5

Line 1220 clears the screen during the calculation and prints an admonition to be patient while the calculation occures. Line 1230 sets the initial number of intervals to two, calls subroutine 1293, which calculates the interval width, DX. Line 1240

AD-A175 327 USE OF THE RADIO SHACK TRS-80 MODEL 100 COMPUTER FOR COMBAT MODELING(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S H CARY SEP 86 2/2

UNCLASSIFIED F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
CROCOPY RESOLUTION TEST CHART

```
1220 CLS:PRINT"":PRINT"              !!Be Patient!!":PRINT""
1230 N=2:GOSUB1293:DX=(XU-XL)/2
1240 FORJ9=1TO6:DX=DX/2:N=N*2
1242 X=XU:GOSUB1296:GOSUB1280:A2(J9,1)=SS*DY
1245 X=XL:GOSUB1296:GOSUB1280:A2(J9,1)=A2(J9,1)+SS*DY
1250 FORJ8=2TON:X=X+DX:GOSUB1296:GOSUB1280
1251 A2(J9,1)=A2(J9,1)+2*SS*DY:NEXTJ8
1252 A2(J9,1)=A2(J9,1)*DX/3
```

Figure 8.5 Integration Calculation.

starts a loop in which the Simpson's Rule intervals are halved at each iteration. That is, in the first iteration YU-YL and XU-XL are divided into four intervals, in the second iteration both y are divided into eight intervals, and so on for six iterations. Line 1240 cuts the interval for X, DX, in half and doubles the number of intervals, N.

Lines 1242-1245 call the subroutines which compute the Simpson Rule sums, SS, at the upper and lower bounds of X. These sums are multiplied by their respective interval widths, DY, and added together into A2(J9,1). Lines 1250-1251 calculate the same summation for values of X between XL and XU at intervals DX and and add the sums to A2(J9,1). Line 1252 multiplies A2(J9,1) by DX/3 to complete the Simpson's Rule approximation of $\iint f(x,y)\,dydx$.

4. **Romberg Extrapolation, Figure 8.6**

```
1255 IFJ9=1THENNEXTJ9
1260 FORJ8=1TOJ9-1
1261 A2(J9,J8+1)=A2(J9,J8)+((A2(J9,J8)-A2(J9-1,J8))/(4^J8-1)):NEXTJ8
```

Figure 8.6 Romberg Extrapolation.

Because the Romberg extrapolations require two numerical approximations, line 1255 skips the extrapolation section after the first iteration, i.e. when J9 = 1. Lines 1260-1261 conduct the Romberg extrapolation as described in the section on Romberg extrapolation in [Ref. 12:pp. 250-276].

5. **Termination and Output, Figure 8.7**

```
1262 T1=A2(J9,J9)-A2(J9,J9-1):IFSGN(T1)*T1-TL>0THENNEXTJ9ELSE1264
1263 PRINT"Tolerance of";TL;"not met after five      extrapolations"
1264 IN=A2(J9,J9)
1265 PRINT"Integral =";:PRINTUSING"######.#####";IN
1266 PRINT"  Actual Tolerance=";:PRINTUSING"##.####";T1*SGN(T1)
1267 SOUND1567,10:SOUND1244,10:SOUND1046,10:SOUND783,20
1268 SOUND1046,10:SOUND783,40
1269 INPUT"Hit Enter To Continue:";Z9:GOTO1205
1275 FORJ7=1TO6:FORJ6=1TOJ7:PRINTUSING"##.###";A2(J7,J6);
1276 NEXTJ6:PRINT"":NEXTJ7:INPUTZ9:RETURN
```

Figure 8.7 Program Termination and Output.

Line 1262 finds the difference, T1, between the last two Romberg extrapolations and compares that difference to the user specified tolerance, TL. If the difference is greater than the tolerance, then the extrapolation is not accurate enough, another numerical integration is conducted with DX and DY halved, and another extrapolation is made. Otherwise, the integral is within tolerance and the program branches to line 1264 to begin the output sequence. If the integral is not within tolerance after six iterations, iterations,[16] the program terminates. Line 1263 prints a message to the screen indicating that the tolerance has not been met. Line 1264 assigns the final value of the integral to IN. Lines 1265 and 1266 print the value of the integral and the actual tolerance,[17] to the screen. Lines 1267 and 1268 play a short tune to cue the operator that the calculation has finished. Line 1269 holds the results on the screen until the operator hits ENTER, cycling the program back to the main menu at line 1205.

6. **Diagnostic Subroutine, Figure 8.8**

Lines 1275-1276 print the A2 matrix. This subroutine can be called in the middle of a calculation to check how far the calculation has progressed. To call the subroutine in the middle of a calculation:

- Hit SHIFT and BREAK together to stop the calculation.
- Enter GOSUB1275.

---

[16]That is, after distances XU-XL and YU-YL have been broken into 128 intervals.

[17]Actual toleranc may be less than the user specified tolerance.

```
1275 FORJ7=1TO6:FORJ6=1TOJ7:PRINTUSING"##.###";A2(J7,J6)
1276 NEXTJ6:PRINT"":NEXTJ7:INPUTZ9:RETURN
```

Figure 8.8 Diagnostic Subroutine, Prints Matrix A2..

- After viewing the matrix, hit ENTER to continue the program.

7. **Simpson's Rule Summation, Figure 8.9**

```
1280 REM Simpson's Rule Sum
1281 Y=YU:GOSUB1285:SS=F:Y=YL:GOSUB1285:SS=SS+F
1282 FORJ5=2TO(N/2):Y=Y+DY:GOSUB1285:SS=SS+4*F:Y=Y+DY:GOSUB1285
1283 SS=SS+2*F:NEXTJ5:Y=Y+DY:GOSUB1285:SS=SS+4*F:RETURN
```

Figure 8.9 Simpson's Rule Summation Subroutine.

Lines 1281-1283 calculate $SS = \sum f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + ,..., 4f_n + f_{n+1}$ where $f_i = f(x,y|x=X)$, $YL \leq y_i \leq YU$, $i = 1,2,3,...,n+1$. n is the number of intervals into which the distance YU-YL has been divided.

8. **F(x,y) to be integrated, Figure 8.10**

```
1285 'f(x,y) to be integrated:
1286 F=1
1288 'X & Y=independent variables.          Hit F8, then F4 When Done.
1289 RETURN
```

Figure 8.10 Subroutine To Calculate f(x,y).

The function to be integrated, f(x,y) is at line 1286.[18] Lines 1285 and 1288 are comments printed to the screen during editing to assist the operator.

[18]An additional line, 1287, may be added if the function is too long for one line.

9. Limits Of Integration, Figure 8.11

```
1290 'Limits of Integration:
1291 'XLOWER/XUPPER are constants.
1292 'YUPPER & YLOWER may be constants or given in terms of X.
1293 XUPPER=1.5707963
1294 XLOWER=0
1295 RETURN
1296 YUPPER=SIN(X)
1297 YLOWER=0
1298 'Hit F8, Then F4 When Done
1299 DY=(YU-YL)/(N+1):RETURN
```

Figure 8.11 Limits Of Integration Subroutines.

Upper and lower limits of integration for X are entered at lines 1293 and 1294 respectively and must be constants. Limits of integration for Y are entered at lines 1296-1297 and may be either constants or functions of X. Line 1299 updates DY. Lines 1290-1292 and 1298 are comments to assist the operator during editing.

F. INTEGRATION SUBROUTINE

The numerical integration program described above is adapted in Figure 8.12 for use as a subroutine. In the subroutine neither f(x,y), the limits of integration, nor the tolerance can not be edited during program execution. All comment lines to facilitate editing have been removed. The subroutine returns IN as the numerical approximation of the integral but does not print IN. The operator must dimension A2(6,6) with the other arrays in the main program and delete line 1201 in the subroutine.

```
1200 'Numberical Integration Subroutine:Steven H. Cary:24 Apr 86
1201 DIMA2(6,6)
1202 TL=.001
1220 CLS:PRINT"":PRINT"      !!Calculating An Integral!!":PRINT""
1230 N=2:GOSUB1293:DX=(XU-XL)/2
1240 FORJ9=1TO6:DX=DX/2:N=N*2
1242 X=XU:GOSUB1296:GOSUB1280:A2(J9,1)=SS*DY
1245 X=XL:GOSUB1296:GOSUB1280:A2(J9,1)=A2(J9,1)+SS*DY
1250 FORJ8=2TON:X=X+DX:GOSUB1296:GOSUB1280
1251 A2(J9,1)=A2(J9,1)+2*SS*DY:NEXTJ8
1252 A2(J9,1)=A2(J9,1)*DX/3
1255 IFJ9=1THENNEXTJ9
1260 FORJ8=1TOJ9-1
1262 A2(J9,J8+1)=A2(J9,J8)+((A2(J9,J8)-A2(J9-1,J8))/(4^J8-1)):NEXTJ8
1263 T1=A2(J9,J9)-A2(J9,J9-1):IFSGN(T1)*T1-TL>0THENNEXTJ9ELSE1266
1264 PRINT"Tolerance of";TL;"not met after five      extrapolations"
1266 IN=A2(J9,J9):RETURN
1275 FORJ7=1TO6:FORJ6=1TOJ7:PRINTUSING"##.###";A2(J7,J6);
1276 NEXTJ6:PRINT"":NEXTJ7:INPUTZ9:RETURN
1280 'Simpson's Rule Sum
1281 Y=YU:GOSUB1285:SS=F:Y=YL:GOSUB1285:SS=SS+F
1282 FORJ5=2TO(N/2):Y=Y+DY:GOSUB1285:SS=SS+4*F:Y=Y+DY:GOSUB1285
1283 SS=SS+2*F:NEXTJ5:Y=Y+DY:GOSUB1285:SS=SS+4*F:RETURN
1286 F=1
1289 RETURN
1293 XUPPER=1
1294 XLOWER=0
1295 RETURN
1296 YUPPER=X
1297 YLOWER=0
1299 DY=(YU-YL)/(N+1):RETURN
```

Figure 8.12 Integration Subroutine.

APPENDIX A

DETECTION SIMULATION PROGRAM LISTING

A complete listing of the Detection Simulation Program is as follows.

```
100 CLS:PRINT"":PRINT"          DETECTION SIMULATION":FORI=1TO400:NEXTI
110 'Input/Initialization
115 OPEN"DSIN"FORINPUTAS1
120 INPUT#1,NS,NP,S1,S2,RH,F1:V1=S1*S1:V2=S2*S2
121 DIMX(NS,5+NP),A2(6,6),T1(3)
125 FORI1=1TO5+NP:INPUT#1,X(1,I1):NEXTI1
126 IFNS=1THEN140
127 IFF1=1THEN132
128 FORI1=2TONS:FORI2=1TO5+NP:INPUT#1,X(I1,I2):NEXTI2:NEXTI1:GOTO140
132 FORI1=2TONS:FORI2=1TO5:INPUT#1,X(I1,I2):NEXTI2:IFNP=0THEN135
134 FORI2=6TO5+NP:X(I1,I2)=X(1,I2):NEXTI2
135 NEXTI1
140 RF=SQR(1-RH^2)
150 DF$="EXP(-((XT-XS)^2+(YT-YS)^2)/(2*X(J2,6)^2))"
200 '*** Simulation Selection Section ***
201 CLS:PRINT"Is the Detection function:"
203 PRINT" 1. Deterministic":PRINT" 2. Probabilistic"
205 INPUT"Enter 1 or 2:";F1
210 CLS:PRINT"Are Sensor Locations:"
212 PRINT" 1. Always At Aim Point":PRINT" 2. Distributed BVN Around Aim Point"
214 INPUT"Enter 1 or 2:";F2
215 IF(F1=2ORF2=2)THENF3=1:GOTO230
220 CLS:PRINT"":PRINT"Is the Calculation:"
222 PRINT" 1=Monte Carlo Simulation":PRINT" 2=Numerical Approximation"
224 INPUT"Enter 1 or 2";F3
230 TIME$="00:00:00":IF F1=1THENGOSUB300ELSEGOSUB500
250 GOTO200
300 'Deterministic Sensor Subroutine
305 IFF3=1THENGOSUB310ELSEGOSUB350
306 RETURN
310 'Monte Carlo of Deterministic Sensor
315 GOSUB900
320 PD=0:FORJ1=1TONR:PRINT.241,"Repetition:";J1:GOSUB600:FORJ2=1TONS
323 IFF2=2THENXS=X(J2,1):YS=X(J2,2):GOTO325
324 GOSUB612
325 T1=SQR((XS-XT)^2+(YS-YT)^2)
330 IFT1<=X(J2,6)THENPD=PD+1:GOTO335
332 IFT1>=X(J2,7)ANDT1<=X(J2,8)THENPD=PD+1:GOTO335
334 NEXTJ2
335 NEXTJ1:PD=PD/NR:GOSUB950:RETURN
350 'Numeric/Deterministic Subroutine
355 PD=0:FORJ2=1TONS:H=X(J2,6):GOSUB1200:PD=PD+IN
356 H=X(J2,8):GOSUB1200:PD=PD+IN
357 H=X(J2,7):GOSUB1200:PD=PD-IN:NEXTJ2
360 GOSUB950:RETURN
500 'Probabilistic Detection Function
502 CLS:PRINT"Default Detection Function Is Carleton."
503 GOSUB1300:GOSUB900
520 PD=0:FORJ1=1TONR:PRINT.241,"Repetition:";J1:GOSUB600:FORJ2=1TONS
521 IFF2=2THENXS=X(J2,1):YS=X(J2,2):GOTO523
522 GOSUB612
523 GOSUB1410:IFRND(1)<=DFTHENPD=PD+1:GOTO526
524 NEXTJ2
526 NEXTJ1:PD=PD/NR
530 GOSUB950:RETURN
600 '***Generate BVN RV***
602 U1=RND(1):U2=RND(1):TE=SQR(-2*LOG(U1))
```

```
604 XT=TE*COS(6.2831853*U2):YT=RH*XT+RF*TE*SIN(6.2831853*U2)
606 XT=XT*S1:YT=YT*S2:RETURN
612 U1=RND(1):U2=RND(1):TE=SQR(-2*LOG(U1))
614 XS=TE*COS(6.2831853*U2):YS=X(J2,5)*XT+(1-X(J2,5)*2)^.5*TE*SIN(6.2831853*U2)
616 XS=X(J2,1)+XS*X(J2,3):YS=X(J2,2)+YS*X(J2,4):RETURN
900 CLS:INPUT"Enter number of repetitions for Monte   Carlo Simulation:";NR
905 RETURN
910 INPUT"Hit ENTER to Continue";Z1:RETURN
950 'Print output
951 SOUND1567,10:SOUND1244,10:SOUND1046,10:SOUND783,20
952 SOUND1046,10:SOUND783,40
953 CLS:PRINT"":PRINT"Calculation Time (HH/MM/SS) = ";TIME$:IFF3=2THEN960
954 PRINT"Select Alpha for Confidence interval:"
955 INPUT" Choices = .1, .05, .01:";AL
956 IFAL=.1THENAL=1.645:GOTO960
957 IFAL=.05THENAL=1.96:GOTO960
958 IFAL=.01THENAL=2.575:GOTO960
959 GOTO954
960 PRINT"*** Estimate of P(Detection) = ";:PRINTUSING" #.#####";PD
961 IFF3=1THEN965
962 PRINT"No Confidence Interval For Numerical    Approximations"
963 GOTO970
965 PRINT"Confidence Interval: (";
966 TE=AL*SQR(PD*(1-PD)/NR):LL=PD-TE:UL=PD+TE:IFUL>1THENUL=1
967 IFLL<0THENLL=0
968 PRINTUSING"##.#####";LL;UL;:PRINT")":GOSUB910
970 'Confetti Approximation
972 PRINT"":INPUT"Confetti approximation? 0=No, 1=Yes:";Z9:IFZ9=0THENRETURN
974 CLS:INPUT"Enter TOTAL lethal area for ALL sensors   in the pattern:";NA
976 TE=NA/(6.283185*S1*S2):TE=1-(1+SQR(2*TE))*EXP(-SQR(2*TE))
977 PRINT"**Confetti Approximation = ";TE:GOSUB910:RETURN
1200 'Numberical Integration Subroutine
1201 D1=6.2831853*S1*S2*RF
1202 TL=.001
1220 CLS:PRINT"":PRINT"      !!Calculating An Integral!!":PRINT""
1230 N=2:GOSUB1293:DY=(YU-YL)/2
1240 FORJ9=1TO6:DY=DY/2:N=N*2
1242 Y=YU:GOSUB1296:GOSUB1280:A2(J9,1)=TS*DX
1245 Y=YL:GOSUB1296:GOSUB1280:A2(J9,1)=A2(J9,1)+TS*DX
1250 FORJ8=2TON:Y=Y+DY:GOSUB1296:GOSUB1280
1251 A2(J9,1)=A2(J9,1)+2*TS*DX:NEXTJ8
1252 A2(J9,1)=A2(J9,1)*DY/2
1255 IFJ9=1THENNEXTJ9
1260 FORJ8=1TOJ9-1
1262 A2(J9,J8+1)=A2(J9,J8)+((A2(J9,J8)-A2(J9-1,J8))/(4^J8-1)):NEXTJ8
1263 T1=A2(J9,J9)-A2(J9,J9-1):IFSGN(T1)*T1-TL>0THENNEXTJ9ELSE1266
1264 PRINT"Tolerance of";TL;"not met after five      extrapolations"
1266 IN=A2(J9,J9):RETURN
1275 FORJ7=1TO6:FORJ6=1TOJ7:PRINTUSING"##.###";A2(J7,J6);
1276 NEXTJ6:PRINT"":NEXTJ7:INPUTZ9:RETURN
1280 REM Trapezoidal Rule Sum
1281 X=XU:GOSUB1286:TS=F:X=XL:GOSUB1286:TS=TS+F
1282 FORJ5=2TON-1:X=X+DX:GOSUB1286:TS=TS+F:NEXTJ5:RETURN
1285 'F(X,Y) to be integrated:
1286 F=X^2/V1-2*RH*X*Y/S1/S2+Y^2/V2
1287 F=(EXP(-F/2/RF^2))/D1:RETURN
1290 'Limits of Integration:
1293 YU=X(J2,2)+H:YL=X(J2,2)-H:RETURN
1296 T3=SQR(H^2-(Y-X(J2,2))^2):XU=X(J2,1)+T3:XL=X(J2,1)-T3:DX=(XU-XL)/N
1297 RETURN
1300 PRINT" -Detection Fn (DF) in terms of XT, YT,"
1302 PRINT"  and Parameters XS, YS, and X(J2,6).... X(J2,5+NP):"
1304 PRINT"  ** DF = ";DF$
1306 PRINT"Hit ENTER For No Change or Enter New...":INPUT" DF = ";DF$
1307 RETURN
1400 'Tokenize DF
1410 B$="DF="+DF$+CHR$(0)
1450 'Tokenize/execute B$
1451 B0=VARPTR(B$):B1=PEEK(B0+1)+256*PEEK(B0+2):CALL1606,0,B1
```

```
1455 CALL2499,0,63105:RETURN
```

APPENDIX B

KALMAN FILTER PROGRAM LISTING

A complete listing of the Kalman Filter Program is as follows.

```
100 CLS:PRINT"*****KALMAN FILTER*****":PRINT"   Input Data Being Read"
110 OPEN"KALIN"FORINPUTAS1:ONERRORGOTO9900
120 INPUT#1,NX,NZ:IFNX<NZTHENMD=NZELSEMD=NX
125 DIMPH(NX,NX),MW(NX),Q(NX,NX),H(NZ,NX),MV(NZ),R1(NZ,NZ),MU(NX),SG(NX,NX)
126 DIMC1(MD,MD),C2(MD,MD),K(NX,NZ)
127 DIMB1(NZ+1,NZ*2)
130 FORI1=1TONX:FORI2=1TONX:INPUT#1,PH(I1,I2):NEXTI2:NEXTI1
132 FORI1=1TONX:INPUT#1,MW(I1):NEXTI1
134 FORI1=1TONX:FORI2=1TONX:INPUT#1,Q(I1,I2):NEXTI2:NEXTI1
136 FORI1=1TONZ:FORI2=1TONX:INPUT#1,H(I1,I2):NEXTI2:NEXTI1
138 FORI1=1TONZ:INPUT#1,MV(I1):NEXTI1
140 FORI1=1TONZ:FORI2=1TONZ:INPUT#1,R1(I1,I2):NEXTI2:NEXTI1
142 FORI1=1TONX:INPUT#1,MU(I1):NEXTI1
144 FORI1=1TONX:FORI2=1TONX:INPUT#1,SG(I1,I2):NEXTI2:NEXTI1
145 CC=0:CLS:PRINT"Initial  SG  As Input Check:":GOSUB532
150 CLS:PRINT"      *****MEASUREMENT BLOCK*****"
160 PRINT"Current  H ":GOSUB540
162 INPUT"Enter New  H ? 1=Yes, 0=No:";Z9:IFZ9=0THEN170
165 'Enter A New  H
167 FORI1=1TONZ:FORI2=1TONX
168 PRINT"Enter Row";I1;", Column";I2;"Of  H :";
169 INPUTH(I1,I2):NEXTI2:PRINT"":NEXTI1
170 'CALC KALMAN GAIN
171 'MULT  SG  H t, INTO C1
172 FORI1=1TONX:FORI2=1TONZ:C1(I1,I2)=0:FORI3=1TONX
174 C1(I1,I2)=(SG(I1,I3)*H(I2,I3))+C1(I1,I2):NEXTI3:NEXTI2:NEXTI1
180 'MULT  H   SG  H t , INTO C2
182 FORI1=1TONZ:FORI2=1TONZ:C2(I1,I2)=0:FORI3=1TONX
184 C2(I1,I2)=(H(I1,I3)*C1(I3,I2))+C2(I1,I2):NEXTI3:NEXTI2:NEXTI1
200 'ADD  R  INTO C2
202 FORI1=1TONZ:FORI2=1TONZ:C2(I1,I2)=C2(I1,I2)+R1(I1,I2)
203 NEXTI2:NEXTI1
210 'INVERT  C2
215 GOSUB9800
220 'MULT  C1  C2  INTO  K
222 FORI1=1TONX:FORI2=1TONZ:K(I1,I2)=0:FORI3=1TONZ
224 K(I1,I2)=(C1(I1,I3)*C2(I3,I2))+K(I1,I2):NEXTI3:NEXTI2:NEXTI1
250 '*****UPDATE MU- TO MU+*****
251 'MULT  H  MU-  INTO  C1
252 FORI1=1TONZ:C1(I1,1)=0:FORI3=1TONX
254 C1(I1,1)=(H(I1,I3)*MU(I3))+C1(I1,1):NEXTI3:NEXTI1
260 'ADD  MV +  H  MU-
262 FORI1=1TONZ:C1(I1,1)=C1(I1,1)+MV(I1):NEXTI1
270 'INPUT A NEW MEASUREMENT
272 CC=CC+1:CLS:PRINT"Measurement #";CC;":"
273 FORI1=1TONZ:PRINT"Enter Element";I1;"Of Measurement:";
274 INPUTZ(I1):NEXTI1
280 'SUBTRACT C1 FROM Z, INTO C1
282 FORI1=1TONZ:C1(I1,1)=Z(I1)-C1(I1,1):NEXTI1
290 'MULT  K  C1  INTO  C2
292 FORI1=1TONX:C2(I1,1)=0:FORI3=1TONZ
294 C2(I1,1)=(K(I1,I3)*C1(I3,1))+C2(I1,1):NEXTI3:NEXTI1
300 'ADD  C2 + MU-  TO UPDATE TO  MU+
302 FORI1=1TONX:MU(I1)=C2(I1,1)+MU(I1):NEXTI1
320 'MULT  K  H  & SUBTR FROM  I , PUT IN C1
322 FORI1=1TONX:FORI2=1TONX:C1(I1,I2)=0:FORI3=1TONZ
324 C1(I1,I2)=(K(I1,I3)*H(I3,I2))+C1(I1,I2):NEXTI3:C1(I1,I2)=-C1(I1,I2)
```

```
326 NEXTI2:NEXTI1
328 FORI1=1TONX:C1(I1,I1)=1+C1(I1,I1):NEXTI1
350 'MULT LAST RESULT BY  SG , INTO C2
352 FORI1=1TONX:FORI2=1TONX:C2(I1,I2)=0:FORI3=1TONX
354 C2(I1,I2)=(C1(I1,I3)*SG(I3,I2))+C2(I1,I2):NEXTI3:NEXTI2:NEXTI1
360 'PUT C2 INTO  SG
362 FORI1=1TONX:FORI2=1TONX:SG(I1,I2)=C2(I1,I2):NEXTI2:NEXTI1
375 CLS:PRINT"Kalman Gain, K(i,j) After"
377 PRINT"Measurement #";CC:GOSUB510
380 CLS:PRINT"Estimate Of System State, MU(i)+ After"
382 PRINT"Measurement #";CC:GOSUB520
385 CLS:PRINT"Estimate Of Covar, SG(i,j)+ After"
387 PRINT"Measurement #";CC:GOSUB530
400 CLS:PRINT"*********MOVEMENT BLOCK*******"
410 'Update MU(CC)+ to MU(CC+1)-
420 'MULT  PH  MU , PUT IN C1
422 FORI1=1TONX:C1(I1,1)=0:FORI3=1TONX
424 C1(I1,1)=(PH(I1,I3)*MU(I3))+C1(I1,1):NEXTI3:NEXTI1
430 'ADD C1+ MW , INTO  MU
432 FORI1=1TONX:MU(I1)=C1(I1,1)+MW(I1):NEXTI1
440 '**UPDATE  SG **
450 'MULT  T  SG , INTO C1
452 FORI1=1TONX:FORI2=1TONX:C1(I1,I2)=0:FORI3=1TONX
454 C1(I1,I2)=(PH(I1,I3)*SG(I3,I2))+C1(I1,I2):NEXTI3:NEXTI2:NEXTI1
460 'MULT  C1  PH t, INTO C2
462 FORI1=1TONX:FORI2=1TONX:C2(I1,I2)=0:FORI3=1TONX
464 C2(I1,I2)=(C1(I1,I3)*PH(I2,I3))+C2(I1,I2):NEXTI3:NEXTI2:NEXTI1
470 'ADD  C2 + Q = SG
472 FORI1=1TONX:FORI2=1TONX:SG(I1,I2)=C2(I1,I2)+Q(I1,I2):NEXTI2:NEXTI1
480 PRINT"Estimate Of System State, MU(I)-"
482 PRINT"Before Measurement #";CC+1:GOSUB520
485 CLS:PRINT"Estimate Of Covar, SG(I,J)- Before"
487 PRINT"Measurement #";CC+1:GOSUB530
490 GOTO160
500 'PRINTING SUBROUTINES
510 'PRINT KALMAN GAIN, K
512 FORI1=1TONX:FORI2=1TONZ:PRINTUSING"######.##";K(I1,I2);:NEXTI2
514 PRINT"":NEXTI1:INPUT"Hit ENTER To Continue:";Z9:RETURN
520 'PRINT  MU
522 FORI1=1TONX:PRINTUSING"######.##";MU(I1);:NEXTI1:PRINT""
524 INPUT"Hit ENTER To Continue:";Z9:RETURN
530 'PRINT COVAR MATRIX,  SG
532 FORI1=1TONX:FORI2=1TONX:PRINTUSING"#####.##";SG(I1,I2);:NEXTI2
534 PRINT"":NEXTI1:INPUT"Hit ENTER To Continue:";Z9:RETURN
540 'PRINT  H
542 FORI1=1TONZ:FORI2=1TONX:PRINTUSING"######.##";H(I1,I2);:NEXTI2
544 PRINT"":NEXTI1:RETURN
550 PRINT" C2 MATRIX:"
552 FORI1=1TOA:FORI2=1TOB:PRINTUSING"######.##";C2(I1,I2);:NEXTI2
554 PRINT"":NEXTI1:INPUT"Hit ENTER To Continue:";Z9:RETURN
9800 'INVERT C2
9815 FORI1=1TONZ:FORI2=1TONZ:B1(I1,I2)=C2(I1,I2):NEXTI2:NEXTI1
9820 FORI1=NZ+1TO2*NZ:FORI2=1TONZ
9822 IFI1=I2+NZTHENB1(I2,I1)=1ELSEB1(I2,I1)=0
9825 NEXTI2:NEXTI1
9830 FORI1=1TONZ
9840 ML=1/B1(I1,I1):FORI3=1TO2*NZ:B1(I1,I3)=B1(I1,I3)*ML:NEXTI3
9842 IFI1=NZTHEN9865
9845 FORI2=I1+1TONZ:IFB1(I2,I1)=0THEN9860
9850 ML=-B1(I2,I1)
9855 FORI3=I1TO2*NZ:B1(I2,I3)=B1(I2,I3)+(ML*B1(I1,I3)):NEXTI3
9860 NEXTI2:NEXTI1
9865 FORI1=NZTO2STEP-1
9870 FORI2=I1-1TO1STEP-1:IFB1(I2,I1)=0THEN9885
9875 ML=-B1(I2,I1)
9880 FORI3=1TO2*NZ:B1(I2,I3)=B1(I2,I3)+(ML*B1(I1,I3)):NEXTI3
9885 NEXTI2:NEXTI1
9890 FORI1=1TONZ:FORI2=1TONZ
9895 C2(I2,I1)=B1(I2,I1+NZ):NEXTI2:NEXTI1
```

```
9897 MI=1:RETURN
9900 IFERL>9700ANDERR=11THENPRINT"!!!ERROR:  C2  Is Not Invertable!!!":END
9905 PRINT"Error Code";ERR;"In l ne";ERL:END
```

APPENDIX C

LANCHESTER SIMULATION PROGRAM LISTING

A complete listing of the Lanchester Simulation Program is as follows.

```
100 'LANCHESTER TIME STEP MODEL
120 MAXFILES=2:OPEN"LANIN"FORINPUTAS1
121 OPEN"LANOUT"FOROUTPUTAS2:INPUT#1,NP,NA,ND
122 IFNA>NDTHENMD=NAELSEMD=ND
124 IFMD<5THENCLS:SF=1
130 DIMAA(NA,ND),BB(ND,NA),AT(NA),DT(ND),AR(NA),DR(ND)
131 DIMQA(2,NA),QD(ND),AB(2,NA),DB(2,ND),SA(NA),SD(ND),OA(NA),OD(ND)
132 'Enter Initial Quantities of Wpns, Break Points And Wpn Types.
134 FORI2=1TONA:INPUT#1,QA(2,I2):SA(I2)=QA(2,I2):OA(I2)=127:NEXTI2
135 FORI2=1TOND:INPUT#1,QD(I2):SD(I2)=QD(I2):OD(I2)=238:NEXTI2
136 FORI2=1TONA:INPUT#1,AB(1,I2):AB(2,I2)=AB(1,I2)*QA(2,I2):NEXTI2
137 FORI2=1TOND:INPUT#1,DB(1,I2):DB(2,I2)=DB(1,I2)*QD(I2):NEXTI2
138 FORI2=1TONA:INPUT#1,AT(I2):NEXTI2:FORI2=1TOND:INPUT#1,DT(I2):NEXTI2
140 TM=0:IFSF=1THENGOSUB600
143 FORI1=1TONP:PRINT#2,"STARTING PHASE";I1
145 'Enter Time Spent In Phase I1 and # of Intervals
146 INPUT#1,TT,NI:DT=TT/NI
150 'Enter Replacement Rates And Attrition Coefficient Matrices
152 FORI2=1TONA:INPUT#1,AR(I2):NEXTI2:FORI2=1TOND:INPUT#1,DR(I2):NEXTI2
154 FORI2=1TONA:FORI3=1TOND:INPUT#1,AA(I2,I3)
156 NEXTI3:NEXTI2
157 FORI2=1TOND:FORI3=1TONA:INPUT#1,BB(I2,I3)
159 NEXTI3:NEXTI2
200 'Fight Phase I1.
202 FORI2=1TONI:TM=TM+DT:PRINT.241,"Phase:";I1;", Increment";I2;"out
of";NI
210 'Fight Time Increment DT.
220 'Update number of attackers
222 FORI3=1TONA:QA(1,I3)=QA(2,I3):NEXTI3:FORI3=1TONA:FORI4=1TOND
223 'IFDT(I4)=1THENQA(2,I3)=QA(2,I3)-AA(I3,I4)*QD(I4)*QA(2,I3)*DT:GOTO226
224 'QA(2,I3)=QA(2,I3)-AA(I3,I4)*QD(I4)*DT
225 QA(2,I3)=QA(2,I3)-AA(I3,I4)*(QA(2,I3)/QD(I4))^DT(I4)*QD(I4)*DT
226 NEXTI4:QA(2,I3)=QA(2,I3)+AR(I3)*DT:IFSF=1THENGOSUB650
227 NEXTI3
230 'Update number of defenders
232 FORI3=1TOND:FORI4=1TONA
233 'IFAT(I4)=1THENQD(I3)=QD(I3)-BB(I3,I4)*QD(I3)*QA(1,I4)*DT:GOTO236
234 'QD(I3)=QD(I3)-BB(I3,I4)*QA(1,I4)*DT
235 QD(I3)=QD(I3)-BB(I3,I4)*(QD(I3)/QA(1,I4))^AT(I4)*QA(1,I4)*DT
236 NEXTI4:QD(I3)=QD(I3)+DR(I3)*DT:IFSF=1THENGOSUB660
237 NEXTI3
240 GOSUB300:NEXTI2
242 IFI1=NPTHENGOSUB350:CLS:PRINT"Output is in file LANOUT.DO.":END
245 PRINT#2,"Status After Phase";I1:GOSUB361:NEXTI1
300 'Check Whether Breakpoint is reached.
320 TF=0:FORI3=1TONA:IFQA(2,I3)>AB(2,I3)THEN325
322 TF=1:PRINT#2,"Attacker Wpn";I3;"Is Below Breakpoint"
323 PRINT#2,"  Bp =";:PRINT#2,USING"####.##";AB(2,I3);
324 PRINT#2,"  Current Level =";:PRINT#2,USING"####.##";QA(2,I3)
325 NEXTI3
335 FORI3=1TOND:IFQD(I3)>DB(2,I3)THEN340
336 TF=1:PRINT#2,"Defender Wpn";I3;"Is Below Breakpoint"
337 PRINT#2,"  Bp =";:PRINT#2,USING"####.##";DB(2,I3);
338 PRINT#2,"  Current Level =";:PRINT#2,USING"####.##";QD(I3)
340 NEXTI3:IFTF=0THENRETURN
350 PRINT#2,"":PRINT#2,"":PRINT#2,"SUMMARY AT END OF BATTLE"
351 PRINT#2,"":PRINT#2,"Time Elapsed During Battle =";
```

```
352 PRINT#2,USING"####.##";TM:PRINT#2,"":GOSUB361
355 CLS:PRINT"Output is in file LANOUT.DO";END
361 PRINT#2," Att Wpn   Breakpoint   Current Level"
363 FORI3=1TONA:PRINT#2,USING"######";I3;
364 PRINT#2,USING"##########.##";AB(2,I3);QA(2,I3):NEXTI3:PRINT#2,""
366 PRINT#2," Def Wpn   Breakpoint   Current Level"
367 FORI3=1TOND:PRINT#2,USING"######";I3;
368 PRINT#2,USING"##########.##";DB(2,I3);QD(I3):NEXTI3:PRINT#2,"":RETURN
600 'Set up output screen
610 PRINT"Wpn #    Attacker            Defender"
620 FORI1=1TOMD:PRINTUSING"##";I1
623 TP=2+I1*8
625 IFI1>NATHEN630
627 LINE(18,TP)-(119,TP+4),1,B:BP=18+INT(100*AB(1,I1))
628 LINE(BP-1,TP+1)-(BP,TP+3),1,B
630 IFI1>NDTHEN635
632 LINE(138,TP)-(239,TP+4),1,B:BP=138+INT(100*DB(1,I1))
633 LINE(BP-1,TP+1)-(BP,TP+3),1,B
635 NEXTI1:RETURN
650 'Update screen output of attackers
653 TP=3+I3*8
655 LINE(OA(I3),TP)-(OA(I3),TP+2),0
656 OA(I3)=18+INT(100*QA(2,I3)/SA(I3))
657 IFOA(I3)>118THENOA(I3)=118:PRINT.(I3*40+2),"*":GOTO659
658 PRINT.I3*40+2," "
659 LINE(OA(I3),TP)-(OA(I3),TP+3),1:RETURN
660 'Update screen output of defenders
663 TP=3+I3*8
665 LINE(OD(I3),TP)-(OD(I3),TP+2),0
666 OD(I3)=138+INT(100*QD(I3)/SD(I3))
667 IFOD(I3)>238THENOA(I3)=238:PRINT.I3*40+22,"*":GOTO669
668 PRINT.I3*40+22," "
669 LINE(OD(I3),TP)-(OD(I3),TP+3),1:RETURN
```

APPENDIX D

GEOMETRIC PROGRAMMING PROGRAM LISTING

A complete listing of the Geometric Programming Program is as follows.

```
100 'Geometric Programming Program
110 OPEN"GEOIN"FORINPUTAS1
120 INPUT#1,NT,NV:K9=NT
122 IFNT-NV<>1THENPRINT"***ERROR: Degree of Difficulty <> 0":END
130 INPUT#1,NC:DIMNT(NC)
140 MN=0:FORI1=0TONC:INPUT#1,NT(I1):IFNT(I1)>MNTHENMN=NT(I1):NEXTI1
143 DIMCT(3,NC,MN),LM(NC),B1(NT+1,NT*2),B2(NT),B3(NT,NV)
145 FORI1=0TONC:FORI2=1TONT(I1):INPUT#1,CT(1,I1,I2):NEXTI2:NEXTI1
150 FORI1=1TONT(0):B1(1,I1)=1:NEXTI1
155 FORI1=NT(0)+1TONT:B1(1,I1)=0:NEXTI1
160 FORI1=2TONT:FORI2=1TONT:INPUT#1,B1(I1,I2):B3(I2,I1-1)=B1(I1,I2)
162 NEXTI2:NEXTI1
170 PRINT"":PRINT"**COMPUTING DELTA'S**"
172 B2(1)=1:FORI1=2TONT:B2(I1)=0:NEXTI1
180 GOSUB9800
200 CLS:I1=1:FORI2=0TONC:FORI3=1TONT(I2):CT(2,I2,I3)=B1(I1,1)
203 PRINT"DELTA(";I2;",";I3;") = ";
204 PRINTUSING"#####.####";CT(2,I2,I3):I1=I1+1:IFI1>5THENGOSUB600
205 NEXTI3:NEXTI2:GOSUB600:CLS
210 PRINT"":PRINT"**COMPUTING OPT FN VALUE**"
212 FORI1=0TONC:LM(I1)=0:FORI2=1TONT(I1):LM(I1)=LM(I1)+CT(2,I1,I2)
214 NEXTI2:NEXTI1
220 FS=1:FORI1=0TONC
222 FORI2=1TONT(I1):FS=FS*(CT(1,I1,I2)/CT(2,I1,I2))^CT(2,I1,I2)
224 NEXTI2:FS=FS*(LM(I1)^LM(I1)):NEXTI1
229 PRINT"":PRINT"F* =";:PRINTUSING"####.###";FS:GOSUB600:CLS
230 'Compute optimal x(n)
232 K9=K9-1
234 FORI1=1TOK9:FORI2=1TOK9:B1(I1,I2)=B3(I1,I2):NEXTI2:NEXTI1
236 CC=1:FORI1=0TONC:FORI2=1TONT(I1)
237 CT(3,I1,I2)=(CT(2,I1,I2)/CT(1,I1,I2)/LM(I1))
238 IFI1=0THENCT(3,I1,I2)=CT(3,I1,I2)*FS
239 B2(CC)=LOG(CT(3,I1,I2)):CC=CC+1:NEXTI2:NEXTI1
242 PRINT"P(m,t)* = opt. value of term t, constr. m, divided by its coefficient."
244 FORI1=0TONC:FORI2=1TONT(I1):PRINT"P(";I1;",";I2;")* =";
246 PRINTUSING"####.####";CT(3,I1,I2):NEXTI2:GOSUB600:NEXTI1:CLS
250 PRINT"":PRINT"** Computing Opt Values Of X(n) **"
260 GOSUB9800
270 CLS:FORI1=1TOK9:PRINT"X*(";I1;") = ";
272 PRINTUSING"######.#####";EXP(B1(I1,1))
273 IFI1>5THENGOSUB600
275 NEXTI1:GOSUB600:END
600 INPUT"** Hit ENTER To Continue: ";Z9:RETURN
9800 'Simultaneous Linear Equation Subroutine: Ax=b
9815 'Invert Matrix A
9820 FORK7=K9+1TO2*K9:FORK8=1TOK9
9822 IFK7=K8+K9THENB1(K8,K7)=1ELSEB1(K8,K7)=0
9825 NEXTK8:NEXTK7
9830 FORK7=1TOK9
9835 IFB1(K7,K7)*SGN(B1(K7,K7))<1E-8THENGOSUB9910
9840 K2=1/B1(K7,K7):FORK6=1TO2*K9:B1(K7,K6)=B1(K7,K6)*K2:NEXTK6
9842 IFK7=K9THEN9865
9845 FORK8=K7+1TOK9:IFB1(K8,K7)=0THEN9860
9850 K2=-B1(K8,K7)
9855 FORK6=K7TO2*K9:B1(K8,K6)=B1(K8,K6)+(K2*B1(K7,K6)):NEXTK6
9860 NEXTK8:NEXTK7
9865 FORK7=K9TO2STEP-1
```

```
9870 FORK8=K7-1TO1STEP-1:IFB1(K8,K7)=0THEN9885
9875 K2=-B1(K8,K7)
9880 FORK6=1TO2*K9:B1(K8,K6)=B1(K8,K6)+(K2*B1(K7,K6)):NEXTK6
9885 NEXTK8:NEXTK7
9890 'Mult A Inverse by b
9894 FORK7=1TOK9:B1(K7,1)=0:FORK8=1TOK9:B1(K7,1)=B1(K7,1)+B1(K7,K8+K9)*B2(K8)
9896 NEXTK8:NEXTK7:RETURN
9900 'Error Routine
9903 IFERL>9700ANDERR=11THENPRINT"!!!ERROR: Matrix Is Not Invertable!!!":END
9905 PRINT"Error Code";ERR;"In Line";ERL:END
9910 'SWITCH ROWS
9915 FORK5=K7+1TOK9:IFB1(K5,K7)*SGN(B1(K5,K7))<1E-8THEN9940
9920 FORK4=1TOK9*2:K3=B1(K7,K4):B1(K7,K4)=B1(K5,K4)
9930 B1(K5,K4)=K3:NEXTK4:RETURN
9940 NEXTK5:PRINT"Error: Matrix Not Invertable":END
```

APPENDIX E

MATRIX ALGEBRA PROGRAM LISTING.

A complete listing of the Matrix Algebra Program is as follows.

```
100 CLS:PRINT"":PRINT"     *** MATRIX ALGEBRA PROGRAM ***":PRINT""
105 PRINT"IS INPUT MATRIX, 'MATIN.DO' IN RAM?":INPUT"  0=NO, 1=YES";FF
107 IFFF=1THENOPEN"MATIN"FORINPUTAS1
300 PRINT"**Enter The Single Largest Dimension of"
305 INPUT"The Largest Matrix To Be Processed: ";K
310 DIMA1(3,K,K),B1(K+1,K*2),R(4),C(4),DET(2):MI=1:OF=1:SF=0
501 CLS:EF=0:PRINT"****MATRIX ALGEBRA PROGRAM MENU****"
504 PRINT" 1. Enter Starting Left Side Matrix"
505 PRINT" 2. Matrix Inversion"
506 PRINT" 3. Matrix Addition":PRINT" 4. Matrix Multiplication"
508 PRINT" 5. Simultaneous Linear Equations"
509 PRINT" 6. Print Current Answer Matrix":PRINT" 7. Other Options";
510 INPUT"   **Enter Number:";CH
512 IFCH=1THENMI=1:Z9=0:GOSUB7006
513 IFCH=2THENMI=1:GOSUB2000
514 IFCH=3THENGOSUB3000
515 IFCH=4THENGOSUB5000
516 IFCH=5THENGOSUB4000
517 IFCH=6THENGOSUB6000
518 IFCH<>7THENGOTO501
520 CLS:PRINT"***MORE CHOICES:":PRINT" 1. Determinant"
524 PRINT" 2. Matrix Integer Exponentiation"
526 PRINT" 3. Store Current Matrix"
530 PRINT" 4. Retrieve Stored Matrix"
531 PRINT" 5. Scalar Multiplication"
532 PRINT" 6. Other Options":INPUT"**Enter Number: "; CH
540 IFCH=1THENMI=1:GOSUB1000
548 IFCH=2THENMI=1:GOSUB7600
549 IFCH=3THENGOSUB8000
550 IFCH=4THENMI=1:GOSUB8200
560 IFCH=5THENMI=1:GOSUB5100
570 GOTO501
700 'PAUSE CONTROL
702 INPUT"** Hit Enter To Continue";Z9:RETURN
800 'INTERMEDIATE MODIFICATIONS
810 PRINT"**Modify The 2nd Matrix?"
812 INPUT"   0=No, 1=Invert, 2=Scalar Multiply: ";Z9
815 IFZ9=0THENRETURN
820 MI=2:IFZ9=1THENGOSUB2000ELSEGOSUB5100
825 GOTO810
1000 'CALC DETERMINANT
1005 IFR(MI)=C(MI)THEN1010
1007 PRINT"ERROR: Number of rows/columns not equal:"
1008 PRINT"       MATRIX IS NOT INVERTABLE!":GOSUB700:EF=1:RETURN
1010 FORI1=1TOR(MI):FORI2=1TOC(MI):B1(I1,I2)=A1(MI,I1,I2):NEXTI2:NEXTI1
1020 DET(MI)=1:FORI1=1TOR(MI)
1021 IFB1(I1,I1)*SGN(B1(I1,I1))<1E-10THENGOSUB1900ELSE1023
1022 IFEF=1THEN1008
1023 DET(MI)=DET(MI)*B1(I1,I1):IFI1=R(MI)THEN1080
1025 FORI3=1TOC(MI):B1(I1,I3)=B1(I1,I3)/B1(I1,I1):NEXTI3
1030 FORI2=I1+1TOR(MI):IFB1(I2,I1)=0THEN1060
1040 FORI3=I1TOC(MI):B1(I2,I3)=B1(I2,I3)-(B1(I2,I1)*B1(I1,I3)):NEXTI3
1060 NEXTI2:NEXTI1
1080 IFOF<>1THENRETURN
1081 PRINT"**Det. Of Matrix ";MI;" Is: ";
1082 PRINTUSING"#####.#####";DET(MI):GOSUB700
1090 RETURN
1900 'SWITCH ROWS
```

```
1910 FORJ=I1+1TOR(MI):IFB1(J,I1)*SGN(B1(J,I1))<1E-10THEN1940
1920 FORJ1=1TOC(MI)*2:TE=B1(I1,J1):B1(I1,J1)=B1(J,J1)
1930 B1(J,J1)=TE:NEXTJ1:GOTO1950
1940 NEXTJ:EF=1:RETURN
1950 DET(MI)=-DET(MI):RETURN
2000 'MATRIX INVERSION
2010 OF=0:GOSUB1000:IFDET(MI)*SGN(DET(MI))>1E-10OREF=1THEN2017
2015 PRINT"*ERROR: Determinant=0. MATRIX NOT INVERTABLE!":GOSUB700:EF=1
2017 IFEF=1THENRETURN
2020 FORI1=1TOR(MI):FORI2=1TOC(MI):B1(I1,I2)=A1(MI,I1,I2):NEXTI2:NEXTI1
2030 FORI1=C(MI)+1TO2*C(MI):FORI2=1TOR(MI)
2032 IFI1=I2+R(MI)THENB1(I2,I1)=1ELSEB1(I2,I1)=0
2035 NEXTI2:NEXTI1
2040 FORI1=1TOC(MI)
2045 IFB1(I1,I1)*SGN(B1(I1,I1))<1E-10THENGOSUB1900ELSE2055
2046 IFEF=1THEN2015
2055 ML=1/B1(I1,I1):FORI3=1TO2*C(MI):B1(I1,I3)=B1(I1,I3)*ML:NEXTI3
2057 IFI1=C(MI)THEN2080
2060 FORI2=I1+1TOR(MI):IFB1(I2,I1)=0THEN2075
2065 ML=-B1(I2,I1)
2070 FORI3=I1TO2*C(MI):B1(I2,I3)=B1(I2,I3)+(ML*B1(I1,I3)):NEXTI3
2075 NEXTI2:NEXTI1
2080 FORI1=C(MI)TO2STEP-1
2100 FORI2=I1-1TO1STEP-1:IFB1(I2,I1)=0THEN2130
2110 ML=-B1(I2,I1)
2120 FORI3=1TO2*C(MI):B1(I2,I3)=B1(I2,I3)+(ML*B1(I1,I3)):NEXTI3
2130 NEXTI2:NEXTI1
2140 FORI1=1TOC(MI):FORI2=1TOR(MI)
2145 A1(MI,I2,I1)=B1(I2,I1+C(MI)):NEXTI2:NEXTI1
2190 OF=1:RETURN
3000 'MATRIX ADDITION
3010 MF=2:MI=2:GOSUB7000:GOSUB800:IFEF=1THENRETURN
3015 FORI1=1TOR(1):FORI2=1TOC(1):A1(1,I1,I2)=A1(1,I1,I2)+A1(2,I1,I2)
3020 NEXTI2:NEXTI1:GOSUB6000:MF=0:RETURN
4000 'SIMULTANEOUS LINEAR EQUATIONS
4010 CLS:PRINT"**Solves Ax=b.  Choices:":PRINT" 1. Enter b Vector."
4012 PRINT" 2. Change An Element In Matrix A."
4013 PRINT" 3. Solve Current Ax=b."
4014 PRINT" 4. Return.":INPUT" * Select A Number: ";CC
4020 IFCC=1GOTO4040
4022 IFCC=2GOTO4050
4024 IFCC=3GOTO4060
4026 IFCC=4THEN RETURN
4035 GOTO 4000
4040 MI=2:R(2)=C(1):C(2)=1:GOSUB7040:GOTO4000
4050 INPUT"**Row, Column Of Matrix A To Be Changed: ";RD,CD
4052 PRINT" - Enter Row";RD;", Column";CD;":";:INPUTA1(1,RD,CD):GOTO4000
4060 MI=1:SF=1:OF=0:GOSUB8000:GOSUB2000
4064 IFEF=0THEN4070
4065 PRINT"*Solution Not Uniquely Determinable":GOSUB700:RETURN
4070 GOSUB5000:CC=0:FORI1=1TOR(2):CC=CC+1:PRINT"x(";I1;") = ";
4075 PRINTUSING"#####.####";B1(I1,1):IFCC>6THENGOSUB700:CC=0
4080 NEXTI1:GOSUB700:SF=0:GOSUB8200:GOTO4000
5000 'MATRIX MULT
5010 MF=1:IF SF=1THEN5020
5015 MI=2:GOSUB7000:GOSUB800:IFEF=1THENRETURN
5020 R(4)=R(1):C(4)=C(2):FORI1=1TOR(4):FORI2=1TOC(4):B1(I1,I2)=0
5022 FORI3=1TOC(1):B1(I1,I2)=A1(1,I1,I3)*A1(2,I3,I2)+B1(I1,I2)
5024 NEXTI3:NEXTI2:NEXTI1:MF=0
5050 IF SF=0THENGOSUB7500:GOSUB6000
5060 RETURN
5100 'SCALER MULT
5110 INPUT"Enter Scalar Multiplier:";SM:FORI1=1TOR(MI):FORI2=1TOC(MI)
5115 A1(MI,I1,I2)=A1(MI,I1,I2)*SM:NEXTI2:NEXTI1:RETURN
6000 'PRINT OUTPUT MATRIX
6010 PRINT" ** Current Answer Matrix:":CC=0:FORI1=1TOR(1):CC=CC+1
6012 FORI2=1TOC(1):PRINTUSING"####.####";A1(1,I1,I2);:NEXTI2:PRINT""
6050 IFCC=5THENGOSUB700:CC=0
6060 NEXTI1:GOSUB700:RETURN
```

```
7000 'MATRIX INPUT
7001 CLS:PRINT"":PRINT"Will This Matrix Be On:"
7002 INPUT" 0=Left, 1=Right";Z9: IFZ9=1THEN7006
7003 R(2)=R(1):C(2)=C(1):FORI1=1TOR(1):FORI2=1TOC(1)
7004 A1(2,I1,I2)=A1(1,I1,I2):NEXTI2:NEXTI1:MI=1
7006 CLS:IFMI=2THEN7008
7007 PRINT"**Choices For Left Hand Matrix":GOTO7009
7008 PRINT"**Choices For Right Hand Matrix:"
7009 PRINT" 1. Enter Matrix From Keyboard"
7010 PRINT" 2. Retrieve Stored Matrix":IFFF<>1THEN7012
7011 IFEOF(1)=0THENPRINT" 3. Enter Matrix From MATIN.DO"
7012 INPUT"**Enter A Number: ";Z9:IFZ9=1THEN7015
7013 IFZ9=3THEN7018
7014 R(MI)=R(3):C(MI)=C(3):GOTO7020
7015 PRINT" **Enter The Rows, Columns"
7017 INPUT"In The Next Matrix: ";R(MI),C(MI):GOTO7020
7018 INPUT#1,R(MI),C(MI)
7020 IF MF<>1THEN7030
7021 IFR(2)=C(1)THEN7030
7022 PRINT"**ERROR: Columns in LEFT MATRIX =";C(1)
7024 PRINT"       Rows In Right Matrix =";R(2)
7026 PRINT"These Must Be Equal For Matrix Mult!!":GOSUB700:EF=1:GOTO7006
7030 IF MF<>2THEN7035
7031 IF(R(1)=R(2)ANDC(1)=C(2))THEN 7035
7032 PRINT"**ERROR:Dimensions For Both Input"
7034 PRINT"Matrices Must Be Equal!!":GOSUB700:EF=1:GOTO7006
7035 IFZ9=2THENGOSUB8200:RETURN
7036 IFZ9=3THEN7050
7037 PRINT" **Fill Matrix Row By Row:":PRINT""
7040 FORI1=1TOR(MI):FORI2=1TOC(MI)
7042 PRINT"-Enter Row";I1;"And Column";I2;":";
7044 INPUTA1(MI,I1,I2):NEXTI2:PRINT"":NEXTI1:RETURN
7050 FORI1=1TOR(MI):FORI2=1TOC(MI):INPUT#1,A1(MI,I1,I2)
7052 NEXTI2:NEXTI1:RETURN
7500 'MAKE ANSWER MATRIX THE FIRST MATRIX FOR THE NEXT OPERATION
7510 R(1)=R(4):C(1)=C(4):FORI1=1TOR(1):FORI2=1TOC(1)
7512 A1(1,I1,I2)=B1(I1,I2):NEXTI2:NEXTI1:RETURN
7600 'MATRIX INTEGER EXPONENTIATION
7610 CLS:PRINT"":INPUT"**Enter Positive Integer Exponent: ";XP
7620 R(2)=R(1):C(2)=C(1):FORI1=1TOR(1):FORI2=1TOC(2)
7622 A1(2,I1,I2)=A1(1,I1,I2):NEXTI2:NEXTI1
7630 SF=1:FOREX=2TOXP:GOSUB5020:GOSUB7500:NEXTEX:GOSUB6000:SF=0:RETURN
8000 'STORE A1(1,,)
8010 R(3)=R(1):C(3)=C(1):FORI1=1TOR(3):FORI2=1TOC(3)
8012 A1(3,I1,I2)=A1(1,I1,I2):NEXTI2:NEXTI1:RETURN
8200 'RETRIEVE THE STORED MATRIX
8210 R(MI)=R(3):C(MI)=C(3):FORI1=1TOR(MI):FORI2=1TOC(MI)
8212 A1(MI,I1,I2)=A1(3,I1,I2):NEXTI2:NEXTI1:RETURN
```

APPENDIX F

NUMERICAL INTEGRATION PROGRAM LISTING

A complete listing of the Numerical Integration Program is as follows.

```
1200 'Numberical Double Integration:Steven H. Cary:24 Apr 86
1201 DIMA2(6,6):TL=.001
1205 CLS:PRINT"":PRINT"       ** Double Integration **"
1206 PRINT"            Romberg Algorithm"
1210 PRINT"0=Edit Function To Be Integrated."
1211 PRINT"1=Edit Limits Of Integration."
1213 PRINT"2=Edit Tolerance; Current Tol.=";TL
1215 PRINT"3=Calculate Integral. ":INPUT"Enter 0, 1, 2, or 3:";Z9
1216 IFZ9=0THENEDIT1285-1288
1217 IFZ9=1THENEDIT1291-1298
1218 IFZ9=2THENINPUT"Tolerance = ";TL:GOTO1205
1220 CLS:PRINT"":PRINT"             !!Be Patient!!":PRINT""
1230 N=2:GOSUB1293:DX=(XU-XL)/2
1240 FORJ9=1TO6:DX=DX/2:N=N*2
1242 X=XU:GOSUB1296:GOSUB1280:A2(J9,1)=SS*DY
1245 X=XL:GOSUB1296:GOSUB1280:A2(J9,1)=A2(J9,1)+SS*DY
1250 FORJ8=2TON:X=X+DX:GOSUB1296:GOSUB1280
1251 A2(J9,1)=A2(J9,1)+2*SS*DY:NEXTJ8
1252 A2(J9,1)=A2(J9,1)*DX/3
1255 IFJ9=1THENNEXTJ9
1260 FORJ8=1TOJ9-1
1261 A2(J9,J8+1)=A2(J9,J8)+((A2(J9,J8)-A2(J9-1,J8))/(4^J8-1)):NEXTJ8
1262 T1=A2(J9,J9)-A2(J9,J9-1):IFSGN(T1)*T1-TL>0THENNEXTJ9ELSE1264
1263 PRINT"Tolerance of";TL;"not met after five      extrapolations"
1264 IN=A2(J9,J9)
1265 PRINT"Integral =";:PRINTUSING"######.#####";IN
1266 PRINT"  Actual Tolerance = ";:PRINTUSING"##.####";T1*SGN(T1)
1267 SOUND1567,10:SOUND1244,10:SOUND1046,10:SOUND783,20
1268 SOUND1046,10:SOUND783,40
1269 INPUT"Hit Enter To Continue:";Z9:GOTO1205
1275 FORJ7=1TO6:FORJ6=1TOJ7:PRINTUSING"##.###";A2(J7,J6);
1276 NEXTJ6:PRINT"":NEXTJ7:INPUTZ9:RETURN
1280 REM Simpson's Rule Sum
1281 Y=YU:GOSUB1285:SS=F:Y=YL:GOSUB1285:SS=SS+F
1282 FORJ5=2TO(N/2):Y=Y+DY:GOSUB1285:SS=SS+4*F:Y=Y+DY:GOSUB1285
1283 SS=SS+2*F:NEXTJ5:Y=Y+DY:GOSUB1285:SS=SS+4*F:RETURN
1285 'F(x,y) to be integrated:
1286 F=1
1288 'X & Y = independent variables.          Hit F8, then F4 When Done.
1289 RETURN
1290 'Limits of Integration:
1291 'XLOWER/XUPPER are constants.
1292 'YUPPER & YLOWER may be constants or given in terms of X.
1293 XUPPER=1.5707963
1294 XLOWER=0
1295 RETURN
1296 YUPPER=SIN(X)
1297 YLOWER=0
1298 'Hit F8, Then F4 When Done
1299 DY=(YU-YL)/(N+1):RETURN
```

LIST OF REFERENCES

1. *TRS-80, Model 100 Owner's Manual*, Tandy Corporation, Ft. Worth, Texas, 1983.

2. Morgan, Christopher L. *Hidden Powers Of The TRS-80 Model 100*, New American Library, 1984.

3. Washburn, Alan, *Search and Detection* Military Applications Section, Operations Research Society of America, 1981.

4. Abramowitz, Milton and Stegun, Irene A., *Handbook of Mathematical Functions With Formulas, Graphs, And Mathematical Tables, Applied Mathematics Series 55*, National Bureau Of Standards, U.S. Government Printing Office, 1964.

5. Washburn, Alan, "Notes On Firing Theory", Naval Postgraduate School Class Notes, 1983.

6. Washburn, Alan, "A Short Introduction To Kalman Filters", Naval Postgraduate School Class Notes, 1985.

7. Kalman, R. E., "A New Approach To Linear Filtering And Prediction Problems", *Trans. ASME - J. Basic Engineer*, 82D, Pages 34-45, 1960.

8. Helmbold, R., "A Modification of Lanchester's Equations" *Operations Research, 13* Operations Research Society of America, 1965.

9. Taylor, James G., *Lanchester Models of Warfare*, Operations Research Society of America, 1983.

10. Reklaitis, G.V., Ravindran, A., and Ragsdell, K.M., *Engineering Optimization - Methods and Applications*, John Wiley and Sons, Inc., 1983.

11. Duffin, Richard J., Peterson, Elmor L., and Zener, Clarence, *Geometric Programming - Theory and Application*, John Wiley and Sons, Inc., 1967.

12. Gerald, Curtis F. and Wheatley, Patrick O., *Applied Numerical Analysis*, Addison Wesley Publishing Company, 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library,Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Professor J. D. Esary, Code 55Ey Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
4.	Associate Professor James N. Eagle, Code 55Er Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
4.	Associate Professor Samuel Parry, Code 55Py Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
5.	CPT Steven H. Cary USA 12221 Eberly Court Wichita, Kansas 67235	2
6.	MAJ Susan Quensel USA 324 N.W. 63rd Street Lawton, Oklahoma 73505	1

END

2-87

DTIC